JPRS-UCC-85-002

4 March 1985

USSR Report

CYBERNETICS, COMPUTERS AND AUTOMATION TECHNOLOGY

FBIS FOREIGN BROADCAST INFORMATION SERVICE

NOTE

JPRS publications contain information primarily from foreign newspapers, periodicals and books, but also from news agency transmissions and broadcasts. Materials from foreign-language sources are translated; those from English-language sources are transcribed or reprinted, with the original phrasing and other characteristics retained.

Headlines, editorial reports, and material enclosed in brackets [] are supplied by JPRS. Processing indicators such as [Text] or [Excerpt] in the first line of each item, or following the last line of a brief, indicate how the original information was processed. Where no processing indicator is given, the information was summarized or extracted.

Unfamiliar names rendered phonetically or transliterated are enclosed in parentheses. Words or names preceded by a question mark and enclosed in parentheses were not clear in the original but have been supplied as appropriate in context. Other unattributed parenthetical notes within the body of an item originate with the source. Times within items are as given by source.

The contents of this publication in no way represent the policies, views or attitudes of the U.S. Government.

PROCUREMENT OF PUBLICATIONS

JPRS publications may be ordered from the National Technical Information Service (NTIS), Springfield, Virginia 22161. In ordering, it is recommended that the JPRS number, title, date and author, if applicable, of publication be cited.

Current JPRS publications are announced in Government Reports Announcements issued semimonthly by the NTIS, and are listed in the Monthly Catalog of U.S. Government Publications issued by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

Correspondence pertaining to matters other than procurement may be addressed to Joint Publications Research Service, 1000 North Glebe Road, Arlington, Virginia 22201.

NOTICE

A new serial entitled WORLDWIDE REPORT: ARMS CONTROL will be published starting in March 1985.

The new report will contain Soviet, European, and other foreign media reportage and commentary on arms control issues, negotiations, agreements, and treaties. Much of the material will be reprinted from the regional FBIS DAILY REPORTS.

U.S. Government consumers may arrange to receive the new report through regular publications distribution channels or by contacting:

FBIS/Liaison and Requirements
P.O. Box 2604
Washington, D.C. 20013

Other consumers may order the report by contacting:

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161

JPRS-UCC-85-002

4 March 1985

USSR REPORT

CYBERNETICS, COMPUTERS AND AUTOMATION TECHNOLOGY

CONTENTS

GENERAL

COMPUTERS AND KNOWLEDGE-PROCESSING

Moscow KOMSOMOL'SKAYA PRAVDA in Russian 6 Nov 84 p 2

[Article by Ye. D'yakova: "Let's Make the Computer an Advisor: The Work of Young Moscow Scientists is Awarded the Prize of the Lenin Komsomol"]

[Text] Remember those excited discussions caused by the first computers? I don't mean the discussions of professionals, but those in the newspapers, during debates, even at high school gatherings. People of all kinds were hotly disputing whether a machine could surpass the human mind. And they would try to show that, despite enormous memory and speed, cybernetics could only handle a limited domain of problems, outside what we call spirituality.

Generally, this was correct. In the past decade the machines have learned to compose music and "paint" the Mona Lisa but, soberly viewed, the range of their problems has remained the same. The computer can "compute" only those processes, about which we know everything, and for which we can construct an exact mathematical model. Unfortunately, such processes in nature are few.

We know very little about the mechanism of the vital phenomena in, say, the human organism, That is why doctors, biologists, geologists and practioners of dozens of "loosely-organized", "non-mathematical", but extremely important fields of knowledge must rely exclusively on their own professional experience, books describing the experience of their predecessors, and experimental data.

Quite a lot, it would seem. But not really.

Thus, for example in the construction of petroleum storage tanks we use tables of coefficients of the metal strength. The table is experimentally obtained; filling in a single square requires the building of a model of the tank and then destroying it. Each new value of the coefficient costs several thousand rubles.

It would appear more easy to compute the coefficients. But, besides our well-known laws of physics and strength of materials, each time dozens or hundreds of factors "interfere" in the matter, which cannot be calculated in the usual way.

But isn't it possible, without going into the actual physics of a particular factor, to simply take account of its effects? It may be so. This is the subject of a special field of cybernetics--the theory of pattern recognition.

For example the computer memory stores detailed descriptions of 200-300 oil wells, each described by dozens of indicators. Using these data, if we know the outlook for 300 wells the computer can reliably predict whether the three hundred and first will be profitable. This method combines the enormous memory and speed of the computer with something that has always been a human attribute--"knowledge of life." But any prediction involves risk. A doctor risks a mistake in diagnosis, experts may err in deciding how to drill a well. How can this risk be minimized in computer predictions?

This question has been answered the best by young Moscow scientists working at the Computer Center of the USSR Academy of Sciences, Igor' Isayev, Dmitriy Kochetkov, and Konstantin Rudakov. All three first heard of the theory of pattern recognition of forms as third-year students beginning their practical training in the laboratory of Yu. I. Zhuravlev, doctor of physical-mathematical sciences and laureate of the Lenin Prize, the founder of the Soviet theory of recognition.

Twelve years passed. The students became Candidates of Science, workers at the Computer Center of the USSR Academy of Sciences, and inventors of a new method of solving recognition problems. The two primary methods of the present day, created by them, have brought the accuracy of solution of the problem up to 90-95% and accelerate the solution process by a factor of 3-5. The young Muscovites have been awarded the Prize of the Lenin Komsomol.

12717
CSO: 1863/85

UDC 681.325.5

THE PROBLEM OF CREATING A STANDARDIZED SERIES OF PERSONAL COMPUTERS

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84
(signed to press 10 Aug 83) pp 3-7

[Article by V. S. Mikhalevich and B. N. Malinovskiy: "The Problem of Creating a Standardized Series of Personal Computers"]

[Text] The chief way for improving the efficiency of our country's national economy on the basis of widely automating and mechanizing labor both in the production and nonproduction realms was determined by decisions of the 26th CPSU Congress and subsequent plenums of the CPSU Central Committee. The implementation of these decisions depends substantially on the level of development and accessibility of computer technology for broad application.

The appearance of personal computers (PEVM), which are the first machines in the history of computer technology of hardware that are designed for individual use, is a turning point in the development of the technological revolution in the area of information science, inasmuch as the personal computer is becoming available to everyone (in terms of cost) and a powerful universal tool that repeatedly improves the productivity of the intellectual labor of specialists with various specializations [1-3].

An analysis of the status of operations in the area of creating and using personal computers abroad confirms the high efficiency of their use and their rapid and broad dissemination in the most diverse realms of human activities that are connected with obtaining and processing information. The sharp competitive struggle between firms and the absence of planned management of the economy with the capitalist method of production led to a large variety of personal computers [4,5].

In our country the potential demand for personal computers is extraordinarily great and their use must yield substantial economic impact. Unfortunately, the development of personal computers that is being conducted at the present time is being poorly coordinated and the personal computers ("Agat," "Veformica," SO-04 and others) that are being produced by industry do not possess a sufficient degree of compatibility.

The planned nature of a socialist national economy and the necessity for optimum use of available resources exclude the need for creating a large quantity of

types of personal computers that are approximately identical in their possibilities. In the present article, an attempt is made to substantiate the necessity and to show ways for developing a standardized series of personal computers, the individual models of which in the technical solutions and software realms would be oriented towards organizing the work places of specialists with various specializations--administrators, researchers, economists, engineers, designers, physicians, teachers, as well as for everyday applications.

The economic impact from using personal computers is formed by virtue of improving productivity in the realm of intellectual labor by means of providing each specialist with a work place that is oriented in the best manner towards his specialized activities and with a transition to ideal, highly efficient manufacturing methods for performing administrative, research and planning operations, public services, and others.

The individual purpose of personal computers demands organization of their massive output, rigidly limits their cost, and requires maximum simplicity also in using small outlays for servicing. In connection with this, it's necessary to stipulate the following with the development of personal computers:

--maximum reduction of cost for a personal computer and compatibility of all models of a series, the design execution of which would meet the requirements of individual use and miniaturization of peripheral devices,

--maximum level of standardization of design and technical solutions and software and methods support, including means that provide for the organization of work places and local networks,

--evolutionary merging of telephone communications networks with local networks on the basis of personal computers and transition to communications channels that provide for simultaneous transmission of data, voice and graphics, and

--creation of a wide selection of applications programs for basic applications of personal computers, as well as network software for operating personal computers in a network with local and distributed data bases. Software must be oriented towards the user who doesn't have experience in operating a computer.

The selection of software and hardware of a series must be open with the aim of providing for constant development and modernization.

Analysis of the architectural and structural solutions of foreign personal computers shows that, in spite of the availability of a large number of diverse models of personal computers that are being produced at the present time by various firms, one can reduce their organization to one well-known main modular architecture.

While considering the experience of creating and operating YeS and SM computers, as well as taking into account the application areas of personal computers, it's advisable to include several modifications of general purpose personal computers (for organizational control, scientific research, engineering calculations, and so forth) and several modifications of domestic purpose

personal computers (for home use, instruction, and others) in comprising the /first priority/ [in boldface] of a standardized series of personal computers.

For example, microprocessor complete sets of the 1810VM86 or 580IK80 series can be used as the basic component base for creating /general purpose/ [in italics] personal computer models of the first stage. The following must be included in a personal computer like this: an 8 or 16-bit central processor; an OZU [main memory] with a capacity from 16K to 1M bytes; a PZU [read-only memory] with a capacity of 16K-64K bytes; a keyboard (60-120 keys); a VZU [external memory device] (1-2 NGMD [flexible magnetic disk unit] or a small size NMD [magnetic disk unit] with a capacity of not less than 10M bytes); an alphanumeric display unit to an ELT [cathode ray tube] (80 x 24) and, in addition in higher-end models, a graphic color display unit (512 x 512); a printer (a wide-format thermal printer in newer models and a small size color ink-jet printer in higher-end ones); a controller for communications with the network and an instrument interface controller; and a power supply unit with a storage battery.

In addition, depending on the problem orientation in the composition of individual models of general purpose personal computers, it's necessary to stipulate an arithmetic expander, a multipurpose unit for debugging software modems, a controller for the input and output of analog signals with a set of modules, a plotter and a plotting board for figuring. All models of a general purpose personal computer must be executed in the form of a single design, however, in the higher-end models of the series of units they will be remote (printer, analog signal input-output unit, schedule output unit, plotting board for figuring). Free connectors will be stipulated as well.

Microprocessor sets of the 580IK80 series can be used as the basic component base for creating /domestic purpose/ [in italics] personal computer models of the first stage. Structural units like the following must be included in an approximate manner as well in the composition of these models: an 8-bit central processor, a main memory with a capacity of 16K-64K bytes, a read-only memory with a capacity of 16K-64K, a keyboard (64-80 keys), an external memory device (KNML [cassette magnetic disk unit] or NGMD), a flat display unit (16 lines) or on a cathode ray tube (80 x 24), and a power supply unit with a storage battery.

Depending on the problem orientation of these models in them, it's necessary to stipulate a printer, a controller for switching on a home black and white and color television set, and a controller for communications with the network. All models will be executed in the form of a single design and only a printer, if needed, must be remote.

The /second stage/ [in boldface] of a standardized series of personal computers must include models that are developed with regard to the prospects of developing a component base and architectural and structural solutions. Right now the characteristics of these models can be indicated only in a hypothetical manner.

It is advisable to develop the following as basic models of a personal computer series of the second stage: (1) a 16 and 32-bit personal computer with flexible architecture on the basis of bit-sliced microprocessor sets with their subsequent implementation on the basis of a single-chip microprocessor with flexible architecture and (2) a 16 and a 32-bit high performance personal computer for using them in multiprocessor systems.

Models of the series of personal computers of the second stage will be distinguished by an improved intelligence level and the ease of using it as a human assistant, particularly to provide for inputting and outputting information in a diverse form and processing it in a natural language in the interactive mode and for implementing the functions of instruction and the deductive generation of conclusions. The load on software will be substantially reduced thanks to the automation of information processing in the form of specifications, the implementation of languages close to the user's language and the appropriate architecture of the computer, and the provision of resources for using the program product of computers of previous generations.

It's advisable to be oriented towards the use of semiconductor multipurpose and made-to-order SBIS [very high-speed integrated circuits] with several hundreds of thousands of gates on a single chip.

The architecture of a personal computer of the second stage must provide for interfacing basic models with special processors and the appropriate peripheral equipment for input, recognition and output of conversational speech; for recognition, processing and output of graphics; as well as for analytical calculations, numerical simulation and processing real-time signals.

The creation of personal computer models of the second stage in a series will demand the execution of a large volume of theoretical research on artificial intelligence, on the development of new computer architectures and distribution systems for processing information with improved resources, and on the creation of a long-range component base for fifth generation computers and high-power automation systems for programming, as well as small-size, high-speed multifunctional peripheral units.

The /main modular principle/ [in italics] of building, which makes it possible to build various configurations of personal computers depending on their specific application, will dominate in personal computer models of the second stage just as in models of the first stage. In this regard, the set of modules will be open, i. e. to allow for adding on and continuous modernization. The second architectural principle--/compatibility/ [in italics] in the internal languages of higher-end models relative to the lower-end ones (including first stage models)--will be observed as well and that will provide for arranging heterogeneous software according to internal languages, as well as for implementing languages that are oriented towards the user.

The necessity of developing and organizing the production of a set of means for building /local networks/ [in boldface] on a personal computer base arises in connection with the large-scale application of personal computers. Thus, already during the development stage of a personal computer series of first

stage, it's necessary to create a set of controllers, adapters and transceivers that provide for building high-speed (5-20M bits per second) local networks on the base of a common bus (of the Ethernet type with a CSMA-CD type protocol) and cyclical networks with sequential transmission of a control marker (Arcnet type) for transmitting data by a coaxial cable and fiber optics circuit, as well as aids for organizing small local networks with simplified protocols and the transmission of data by different two-wire circuits (including dedicated telephone channels) with a rate of 50-500K bits per second. A series of modems that are compatible with the personal computer in terms of design must be developed in addition for output to the global networks.

We are faced with developing network modules in designing personal computers, standardizing interunit interfaces, and determining the types of necessary coaxial cables, fiber optics circuits and interfaces with them.

It is advisable to create a multipurpose microprocessor communications junction that provides multinetwork functions for a personal computer series of the second stage.

A junction like this must include a microprocessor (from the personal computer series), a set of systems-wide and applied programming modules that adapt the communications junction to different networks, sources and receivers of the most diverse information (data, voice, graphics), interface modules and modules for joining with data transmission channels.

The software of a network must include the following: a multijob, real-time operating system that consists of a nucleus, low-level standardized network service modules, protocol modules that differ from the standard, and interface protocol modules (foreign analogs are CP/Net and UNIX). Modules for supporting the protocols of the three upper levels of network service must be included as well in the software of a personal computer network.

For providing the local networks of personal computers with a centralized means of storing information, it's advisable to include units for storing and servicing data banks in the composition of hardware for a personal computer series of the second stage.

The orientation of personal computers towards the mass user and the diversity of their applications determine the distinguishing features of the approach to developing their /software/ [in boldface], although in principle its structure differs very little from the software structure of multipurpose computers.

/Systems software/ [in italics] of a personal computer series includes a set of programming modules that implement the organization and execution of functions of applied tasks in a specific personal computer configuration (OS) [operating system] and the programming modules of translators, assemblers, compilers and interpreters that provide for the effective interaction of users with the personal computers in high-level languages.

It's possible to single out three types of operating systems that are being used in personal computers--single-programming, multiprogramming and real-time.

At the present time the CP/M-80 and CP/M-86, which are distinguished by conceptual simplicity, reliability and ease of assimilation and working with them, are the most widely disseminated operating systems. The MP/M-80, MP/M-86 and CP/Net systems, which have a file structure that is compatible with the file structure of the CP/M system, are the evolution of them and that makes it possible to use software that was created earlier. The MP/M system is a multiterminal version of CP/M that provides for more efficient use of such personal computer resources as the processor, main memory and external memory. The CP/Net system is oriented towards creating the local networks of personal computers and it has the means for generating various exchange protocols between personal computers. It's necessary as well to consider the UNIX, XENIX and OASIS computer-independent systems as prospective operating systems.

BASIC, Structural BASIC, PASCAL, FORTRAN, ADA and others have obtained wide dissemination in personal computers as high-level languages. It's necessary to note a number of tendencies that are connected with the expanded use in personal computers of structural languages which are oriented towards a specific application. In addition to the complete set of compilers, practically all operating systems include filing systems and control systems of relational data bases (DATASTAR, BASE-II, SUPERSOFT), and for all categories of users that substantially simplifies the solution of tasks for creating and updating large information files of a different structure.

The /applied software/ [in italics] of personal computers consists of a set of problem-oriented modules (applied program packs) that provide their problem orientation.

Since personal computers are oriented towards the mass user, the demands, which are similar to the demands on a consumer item, are on the applied software of the personal computers: it must be simple in accessing, accessible to the unskilled user, and always ready for use.

In many ways, the effectiveness of using personal computers and the automated work places that are being created on their basis is determined by the possibilities of the /peripheral units/ [in boldface], and first of all the information display devices--the display units.

The demands of users on the display units are extraordinarily diverse--from displaying simple character information to analysing and synthesizing complex graphic scenarios. Therefore, the creation of a multipurpose display unit that satisfies the widest range of personal computer users is not advisable because of the high cost, complexity and notorious excessiveness of its technical implementation.

It's necessary to develop a series of video terminals with a reorganizable structure that are modularly expandable and compatible in terms of software, hardware and design. The characteristics of video terminals like this can be changed in accordance with the demands of users during the creation of automated work places depending on the specific practical task.

Multipurpose coding units of the plotting board type, which provide the following, are necessary for the input of graphic information to personal computers:

--input of symbolic information with an unlimited alphabet (an open set of symbols can be obtained through a simple change of alphabets in the unit's operating field),

--input of symbolic information by means of manual selection of graphics components on a display screen, i. e. the user actually has available a keyboard that is being received in a programmed manner (numerous television video games are built in this kind of information input), and

--input of manuscript and graphic information.

When creating personal computers of the second stage, it's advisable to supplement the multipurpose symbolic and graphic input with a voice input of information, the digital coding of which can be accomplished by a microphone with a digital output that is adaptive in sensitivity. Technically, the task of digitally coding a wide range of acoustical signals can be solved by virtue of expanding the set of discrete components that are designed for processing analog signals.

While taking into consideration the great variety of automation tasks in which personal computers can be used, a standardized system of hardware must be developed for interfacing personal computers with entities with a main modular organization that provides for the building from standardized units of measuring and (or) executive subsystems that are diverse in composition and technical characteristics. In this regard, it's necessary to envisage the possibility of unlimited extension and improvement of the products list of functional units without changing the structure, organization and design of the over-all portion of the basic modification.

Automated work places are being created on the base of personal computers:

(1) in the area of administrative management for the manager of an enterprise (institution, organization), an economist, a secretary-abstractor, and workers of personnel departments and other departments and services,

(2) in the area of scientific research and development for conducting scientific, theoretical and experimental research, planning and design operations, as well as for automating reference information operations and formalizing research results, i. e. for editing, storing and printing out texts, drawing sketches and schedules, preparing technical specifications, and so forth,

(3) with the aim of instruction for students of schools, tekhnikums, VUZ's and various additional training courses, as well as for teachers and developers of instructional courses,

(4) in the area of medicine for solving tasks of automating the diagnosis of patients, organizing reference services and managing paperless medical case records, and

(5) in the area of everyday applications for organizing television video games, controlling household appliances, obtaining information from an urban data bank, and managing a family information data base, home instruction, and so forth.

Already during the creation of a standardized series of personal computers of the first stage, a personal computer of microprocessor development (PEVM-R) will be manufactured for independent debugging of software and hardware units of microprocessor equipment, its comprehensive debugging, as well as for monitoring, preventive, and diagnostic operations. During independent debugging, the software of the PEVM-R will provide for simulation in time, which is close to the actual one, of all hardware assemblies of the microprocessor equipment that is being developed and the information streams that are circulating through the input-output ports; the assignment by the operator of the initial condition of the memory cells and program-accessible assemblies; the test run of a program until execution of the prescribed conditions and output of the contents of the memory cells; and so forth. Depending on the task that is being solved, the PEVM-R will simulate the missing hardware assemblies of the microprocessor equipment, while assigning to the operator all the enumerated resources.

Thus, the creation of a standardized series of personal computers will exert a substantial influence on the further development of computer technology. The functions of hardware of VTs [computer centers] will be changed substantially: on the one hand, a considerable volume of computations, with which personal computers will be loaded, will be removed from them; and, on the other hand, the possibility and necessity are appearing for a problem orientation of hardware towards the most complex tasks, the solution of which for the time being is too much for modern computer technology. The appearance of personal computers and local networks on their basis will accelerate the merging process of computer technology and communications into a single system for processing and transmitting information [6].

BIBLIOGRAPHY

1. Romanov, A. K., "Microprocessor Technology and Automation of the National Economy," MIKROPROTSESSORNYYE SREDSTVA I SISTEMY [Microprocessors and Systems], No 1, 1984, pp 3-6.

2. Naumov, B. N. and Giglavyy, A. V., "Microprocessor Manufacturing Methods--The Basis of Long-Range Computers for Mass Application," MIKROPROTSESSORNYYE SREDSTVA I SISTEMY, No 1, 1984, pp 7-10.

3. Proleyko, V. M., "Microprocessor Aids of Computer Technology and Their Application," MIKROPROTSESSORNYYE SREDSTVA I SISTEMY, No 1, 1984, pp 11-16.

4. Gromov, G. R., "Personal Computations--A New Stage of Information Technology," MIKROPROTSESSORNYYE SREDSTVA I SISTEMY, No 1, 1984, pp 37-50.

5. Yakovlev, Yu. S., Novikov, B. V. and Nesterenko, N. V., "Features of the Application and Architectural and Structural Organization of Personal Computers," (review), UPRAVLYAYUSHCHIYE SISTEMY I MASHINY, No 5, 1984, pp 7-12.

6. Glushkov, V. M., "The Bases of Paperless Information Science," Moscow, NAUKA, 1982, 552 pages.

9889
CSO: 1863/69

MICROCOMPUTER REPAIR SERVICE EXCORIATED

Kishinev SOVETSKAYA MOLDAVIYA in Russian 5 Oct 84 p 4

[Article by Ye. Zamura: "The Computer For-Interior Decoration"]

[Text] The story we are about to tell has become overgrown with a thick dossier of incoming and outgoing documents.

It all began when the computer complex received in November of last year by the Structural Mechanics Department of the Kishinev Institute of Agriculture imeni Frunze proved to be faulty.

"The display doesn't work!" announced the head of the Kazan plant, responding to the alarm. "I can't get it to work. Take it to Fryazino, where it was manufactured."

The answer received from Fryazino was as short as a telegram: Consult Kazan for all problems in the use of computer complexes.

After a long and fruitless correspondence, department head Yu. Mongolov finally discovered the following: Since the peripherals of the computer system are obtained by the Kazan plant as finished articles by agreement with the Fryazino plant, it is not appropriate to fix them. But appealing to Fryazino also proved useless: after the article delivered to the assembly factory has passed inspection on receipt, the supplier is no longer responsible.

It would need an electronic brain to solve this conundrum. But the machine was sick and could offer no assistance. It was then that the department, in despair at untangling this Gordian knot, sent a letter requesting help to the Kazan Oblast Party Committee.

The system for which a considerable sum had been paid was finally fixed as a special dispensation after a half year of idleness. But the favorable ending of this story is hardly grounds for optimism. Today, two of the five computer complexes at the institute need repairs, but there is no one and no place to do this.

Microcomputers with peripherals are widely used in various branches of science and industry at the present day. The Kishinev store "Instruments" has sold

more than 20 of these in the past year alone. What is the fate of these devices? Unfortunately, many of them are nonfunctional today, as at the agricultural institute: the complex fragile systems require constant qualified maintenance.

"Our Elektronika DZ-28 has been down for one and a half years" complains A. El'kin, dean of the department of semiconductor and microelectronic devices at the Kishinev Polytechnical Institute imeni Lazo. "The system costs 25,000 rubles, its service life is 10 years. Thus, we have already irretrievably lost nearly 4,000 rubles.

But there is still more in the matter of losses. At the Department of Mechanization of the Moldavian Scientific Research Institute for Viticulture and Winemaking, for example, a computer complex based on the Elektronika DZ-28 is used for seasonal field studies. If the computer fails, as happened this summer, the information must be processed "by hand". Work that the computer can do in an hour takes around 1-2 months for a person. Which means that the continued testing of a new tractor for grape planting must wait until the following season.

But who fixes a faulty computer? Specialists in maintenance of the microcomputers of the Elektronika DZ-28 system are located only in Kazan, Smolensk and Yerevan. Previously, there used to be a travelling repair brigade at one of the Yerevan plants, for example. Requests from users in nearby rayons and oblasts would pile up over several months until a specialist was sent out to "service" the entire region.

Today, "Mohammed comes to the mountain": giving up hope in arrival of repairmen, the members of the scientific research institutes of Moldavia simply load the packaged computer complex (weighing, incidentally, more than 20 kilograms) onto their back in a knapsack and, with the blessing of their superiors, head out for the manufacturer. If the capricious machine is not damaged by jostling on the way back, the repair problem is solved--until the next breakdown, of course.

Incidentally, the consumer who has learned what to expect from repair of electronics is seeking other ways out of the dilemma. Who, for example, would advertise the fact that two out of three expensive systems purchased are intended...for spare parts? Especially since no one is presently checking on the utilization coefficient of the hardware.

But just how and where does one rehabilitate an electronic brain? Faced with this question, a person unfamiliar with problems of servicing of microcomputers would first consult the Kishinev plant for repair of computer technology.

"But to no avail!" chorus the director of the factory G. Benzar' and his assistant V. Novitskiy. "We have neither the equipment nor the specialists needed for repairs. Thus, according to instructions, the enterprises manufacturing such equipment are the ones that should maintain it. But our assortment covers the hardware of the system TsSU SSSR: paper punching and keyboard machines, calculators...."

Still and all, there may be a solution. Regional centers must be created for maintenance and repairs of microcomputers of all systems. In regard to Moldavia, why not organize such a center on the premises of the Kishinev factory for repair of computer technology?

And the obstacles mentioned by those in charge of the factory are not all that insuperable: it is not likely that any enterprise manufacturing a microcomputer will refuse to help with the necessary repair equipment or training of specialists.

The Gosplan of the Moldavian SSR should seriously study this problem, equally important to all the offices and organizations of the republic making broad use of microelectronics.

The mini-computer is not meant to be an interior decoration.

12717
CSO: 1863/85

HARDWARE

UDC 681.325.5-181.48

USE OF FUNCTION PROCESSORS IN MULTIPROCESSOR SYSTEMS

Leningrad IZVESTIYA VYSSHIKH UCHEBNYKH ZAVEDENIY: PRIBOROSTROYENIYE
in Russian Vol 27, No 5, May 84 (manuscript received 4 Oct 83) pp 39-42

[Article by A. V. Anisimov, A. V. Kraynikov, B. A. Kurdikov and V. B. Smolov, Leningrad Electrical Engineering Institute imeni V. I. Ul'yanov (Lenin)]

[Text] The organization of multiple-variable function computation in multiprocessor systems is analyzed. Transmission of initial data to the processor solving the task at hand is proposed via a memory register array whose functions are similar to those of a list of formal subprogram parameters. The advisability of incorporating an integral matrix multiplier in the processor used for computing multiple-variable functions is indicated.

The use of multiprocessor systems is an effective means of reducing calculation times through parallel operation of separate processors. The effectiveness of such systems is determined by the degree to which their architecture corresponds to the characteristics of the problems to be solved. Multiprocessor systems are used in the solution of an important class of problems, those concerned with the modeling of complex dynamic systems. This class of problems is characterized by the presence of a large number of operations involving the computation of multiple-variable functions. In aircraft modeling for example, these operations take up some 40-60 percent of the total computing time. With this in mind, it seems advisable to incorporate function expanders dedicated to the computation of multiple-variable functions. Overall system efficiency can be increased substantially by ensuring parallel operation of the expanders and central processor. For this, the questions of (1) organizing initial data transmission to the function expanders and receipt of computation results, (2) ensuring parallel central processor/function expander operation, and (3) synchronizing the calculation processes carried out by the function expanders and central processor must be resolved.

Tasks are executed by a computer configuration which is illustrated in Fig. 1. This architecture includes a SM EVM (International System of Small Computers), equipment (processor, memory and peripheral units [PU]) connected

to its common bus. The function expansion unit is connected through a common bus segment, formed as a typical interface segmentation device (SGI). Functional processors designed to solve systems of linear and nonlinear equations with variable coefficients are connected to this segment. The number of such segments can be increased if necessary. Because the interface is partitioned into segments, its basic portion can be relieved of interchanges required for operation of the function expanders.

1 Common bus
2 Memory, SM EVM
3 Processor SM EVM
4 Interface segmentation device
5 Peripheral unit (PU)
6 Peripheral unit (PU)
7 Common bus segment
8 Local memory
9 Multiple-variable function (1)
10 Function expander (2)
11 Function expander (end)

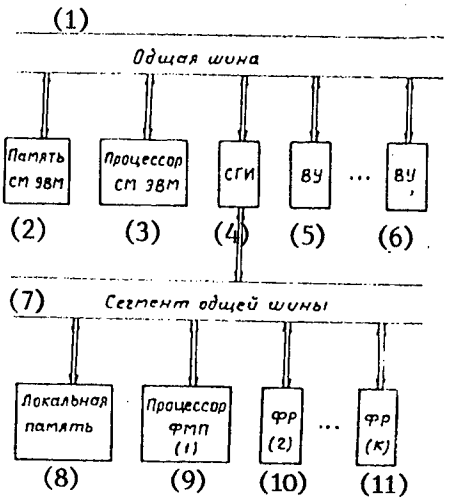


Fig. 1

We will illustrate the special characteristics involved in the use of function expanders using the example of a multiple-variable function (MVF) processor, which is most interesting from the standpoint of organizing interaction with controlling subsystems. Table interpolation methods are used in order to compute functions in the MVF processor. Here various functions are achieved by a single interpolation program by means of shifts in a nodal point table. Nodal point tables corresponding to the functions reproduced are stored in the segment's local memory. Several interpolation formulas, selectable upon formulation of the application program, are included in the MVF processor to ensure effective execution of functions withthheir special characteristics. Let us examine the process of initiating multiple-variable function calculations. It is assumed that nodal point tables for all functions which might be encountered in modeling the dynamic system have been previously loaded into the segment's local memory. Formulas of the following type are used for calculating these functions:

$$f(x_i+ph,\ y_j+qk)=\frac{q(q-1)}{2}f(x_i,\ y_j-k)+$$
$$+\frac{p(p-1)}{2}f(x_i-h,\ y_j)+(1-pq-p^2-q^2)f(x_i,\ y_j)+$$
$$+\frac{p(p-2q+1)}{2}f(x_i+h,\ y_j)+\frac{q(q-2p+1)}{2}f(x_i,\ y_j+k)+$$
$$+pqf(x_i+h,\ y_j+k),$$

where x_i, y_j are nodal values for the arguments; h, k are separations between nodal points; and p, q are standardized values for the arguments.

The above formula corresponds to uniformly distributed interpolation nodes. This allows the use of basing methods in the search for interpolation nodes in local memory. Arguments X,Y, consisting of an address portion XA, YA (kx, ky of high order bits) and normalized arguments p,q (n - kx, n -ky of low order bits) are used to determine the location, relative to the base, of arrays containing nodal points for the function. Depending on the type of function, the size of kx, ky can be a variable, giving rise to the necessity for a clear indication of the length of the address portion for each argument in each of the functions to be calculated in the MVF processor. Information on the size of kx,ky is placed in the table header for the designated function.

To start the MVF processor, the central processor loads its control registers with a control word consisting of an operation code and an address (Fig. 2). First the arguments must be transferred to a memory register established for each function expander. The operation code ($КО_р$) selects one of the interpolation algorithms and the address specifies a function table. This completes the multiple-variable function computation startup sequence and the MVF processor itself is responsible for further execution, including converting arguments, reading nodal points from segment local memory and computing functions according to the interpolation formula.

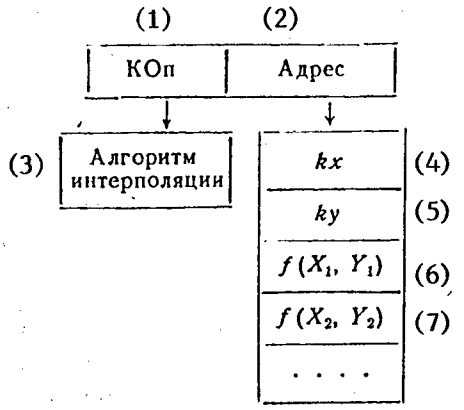


Fig. 2

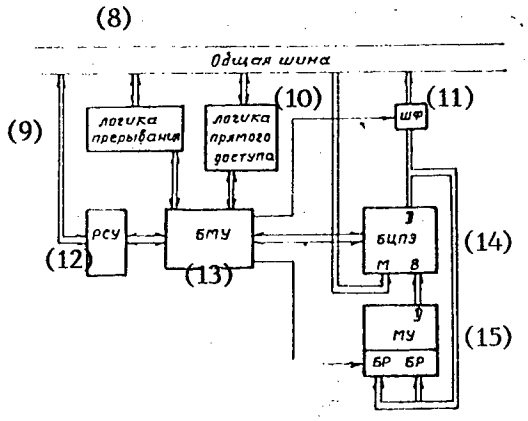


Fig. 3

1 Operation code
2 Address
3 Interpolation algorithm
4-7 (stet)
8 Common bus
9 Interrupt logic
10 Direct access logic
11 Bus driver
12 Status and control register (SCR)
13 Microprogram control unit (MCU)
14 Central processing unit (CPU)
15 Array multiplier (AM)
Buffer register (BR)
Buffer register (BR)

D = Data
M = Array
B = Input

Let us examine the MVF processor's operation according to the block diagram provided in Fig. 3. When calculation begins, the base address of the table for the function undergoing processing is loaded into the address register, which enters into the direct access logic. The start of calculation initiates the shift of the operation code to the status and control register (SCR). To facilitate computation according to the interpolation formula contained in the MVF processor, an array multiplier (AM) is included in addition to the microprogram control unit (MCU) and central processing unit (CPU). To carry out computations in the array multiplier, factors from the CPU data output are transferred to buffer registers (BR). Because the CPU data output port is also connected to the common bus, a bus driver (BD) is included in the MVF processor in order to assure that this operation occurs regardless of bus status.

The MVF processor has direct memory access in order to obtain table data from the segment local memory during the calculation process. Upon completing the function calculation, the MVF processor transfers the results into a memory area established for each function expander and initiates the CPU interrupt required to synchronize computational processes. Several function expanders can operate in parallel. Any conflicts which may arise are resolved by the interface segment. The assignment of priorities among the function expanders is determined by the frequency with which they are accessed and depends on the frequency proportion of operations with specific algorithms. For example, a high specific share of function calculation operations gives rise to higher priority for the MVF processor.

A characteristic of the organization of recourse to a functional processor analyzed is its similarity to subprogram access. The transmission of arguments and results via a fixed memory area is a variation of the implementation in which functions are calculated according to a list of real subprogram parameters. In this variation the main program, executed by the CPU, accesses a function expander--the MVF processor. This reduces the time of task allocation among the function expanders and assures their capacity to work independently within the limits of the common bus segment.

The paper is recommended by the Department of Computing Engineering.

BIBLIOGRAPHY

"Malye EVM i ikh primeneniye" [Small Electronic Computers and Their Application]. B. N. Naumova, editor. Moscow: Statistika, 1980--231 pp.

12746
CSO: 1863/174

UDC 681.325

BIPOLAR NONSWITCHING LOGIC ARRAYS--UNIVERSAL COMPONENTS OF FUNCTION EXPANDERS FOR COMPUTATION OF ELEMENTARY FUNCTIONS

Leningrad IZVESTIYA VYSSHIKH UCHEBNYKH ZAVEDENIY: PRIBOROSTROYENIYE in Russian Vol 27, No 5, May 84 (manuscript received 27 Jan 83) pp 48-53

[Article by V. D. Baykov, A. I. Krys', Leningrad Electrical Engineering Institute imeni V. I. Ul'yanov (Lenin)]

[Text] We examine two versions of the design of function expanders intended for "digit by digit" calculation of elementary functions in Series K1804 microprocessor devices (MD) and in bipolar nonswitching logic arrays (NLA). Specific rated values are given for equipment, time, energy and cost factors in the design and production of both types of function expanders. The advantages of functional processor production based on the NLA variation are indicated in comparison to the MD version.

Continual improvement in the architecture and structure of microprocessor large-scale integrated circuits and the growth in the number of microprocessor devices gives the impression that microprocessors are becoming universal basic components capable of rapidly covering shortfalls in a range of specialized computers, such as function expanders designed for the calculation of elementary functions.

We will consider the potential possibilities of the functional expanders designed in two variations--on the elements of one of the most promising 4-bit slice series K1804 microprocessor devices (MD) [1], and on the basis of bipolar nonswitching logic arrays (NLA) [2]. The validity of such a comparison is explained in that MD components and NLA chips are produced by similar technologies and are capable of nearly identical high-speed operation. A modified "digit by digit" iteration method with changing sign-permanent sign increments was chosen as an effective computation method for the comparison of both versions [3]. As an example we will examine function expanders used in logarithmic mathematics because they are the most complicated and claim universality of application, i.e. they have the capacity to compute other functions without requiring configuration changes.

Recurrent correlations chosen for microprogram- and hardware-calculated logarithmic functions take the form

$$(1)\quad \text{Этап 1}\begin{cases} X_{i+1} = X_i + \xi_i \cdot 2^{-(i+1)} Y_i, \\ \xi_{i+1} = \begin{cases} -1, & \text{если } X_{i+1} \geqslant 1, \\ +1, & \text{если } X_{i+1} < 1, \end{cases} \\ Y_{i+1} = X_{i+1}(\xi_{i+1}+1)\cdot 2^{-1} - Y_i(\xi_{i+1}-1)\cdot 2^{-1}, \\ q_{i+1} = \begin{cases} 0, & \text{если } \xi_{i+1} = -1, \\ 1, & \text{если } \xi_{i+1} = +1, \end{cases} \end{cases}$$

$$(2)\quad \text{Этап 2}\ \{\Theta_{i+1} = \Theta_i + \log_B(1 + q_{i+1}\cdot 2^{-(i+1)}),$$

1 Stage 1 если = if 2 stage 2

where i=0,1,2..., (n-1); X_i, Y_i are the current values of argument X; ξ_i, q_i are the values of the controlling operator corresponding to the sign change or sign maintenance processes; and Θ_i is the current value of the function being computed.

Initial conditions: $X_o = X^*$, $Y_o = 0$, $\xi_o = 1$, $q_o = 1$, $\Theta_o = 0$, X[0,1;1].

Result $\Theta_n = \log_b X$

In order to avoid limiting the generality of research conducted, in order to compare the expenditures required for the design of function expanders based on MDs and those based on NLA hardware, there is no need to write a microprogram to compute $\log_b X$ function in series K1804 microprocessor code. It is sufficient to examine the basic microprogram steps giving primary weight to time and comparing them in terms of the number and length of steps. The results of microprocessor analysis are given in Table 1. The number of steps needed to calculate the $\log_b X$ function is determined according to the formula

$$N_T = N_{MK} + (m-1)(N_{MK} - N_{NU} - N_{norm}),$$

where N_{MK} is the number of microcommands; m is the operand word length, N_{Nu} is the number of initial conditions; and N_{Norm} is the number of normalization steps. The following conditions were established for the $\log_b X$ function according to recurrent correlations (1): $N_{nu} = 7$, $N_{norm} = m-1$.

(2) *Таблица 1*

(1) Основные шаги микропрограммной реализации	Число тактов в зависимости от разрядности m операндов						
	4	8	12	16	24	32	64
(3) 1. Запись начальных условий, ввод переменной X	7	7	7	7	7	7	7
(4) 2. Нормализация переменной X $(X \Rightarrow X^*)$	3	7	11	15	23	31	63
(5) 3. Сдвиг Y_l на 1 разряд, Сложение X_l с Y_l	2	2	2	2	2	2	2
(6) 4. Сравнение $(X_{l+1} - 1) \geqslant 0$, Вычитание, Пустой такт для присвоения. Присвоение значения ξ_{l+1}	3	3	3	3	3	3	3
Либо сложение и сдвиг, Умножение на m-разрядное число							
Либо вычитание и сдвиг, Умножение на m-разрядное число	2	2	2	2	2	2	2
(7) 5. Присвоение значения q_{l+1}, Пустой такт для определения текущего значения θ_{l+1}	2	2	2	2	2	2	2
(8) 6. Присвоение $\theta_{l+1} = \theta_l$, $\theta_{l+1} = \theta_l + \log_B (1 + 2^{-l})$	2	2	2	2	2	2	2
(9) 7. Переход на следующую итерацию	1	1	1	1	1	1	1

Table 1

1		Basic steps for the microprocessor version
2		Number of steps as a function of operand word length m
3	1.	Write initial conditions, Input variable X
4	2.	Normalize variable X (X=>X*)
5	3.	Shift Y_i one bit, Addition of X_i and Y_i
6	4.	Compare $(X_{i+1} - 1) \geq 0$
		Subtraction
		Idle step for assignment.
		Assign value ξ_{i+1}
		or add and shift
		Multiply by m-bit quantity
		or subtract and shift
		Multiply by m-bit quantity
7	5.	Assign value q_{i+1}
		Idle step to determine current value of Θ_{i+1}
8	6.	Assign $\Theta_{i+1}=\Theta_i$
9	7.	Jump to next iteration

Microprogram execution time is determined by the formula

$$T_{\Sigma} = T_T \cdot N_T,$$

where T_T is the step duration. For the K1804 microprocessor set, step duration is approximately equal to the execution time for one microcommand T_{mk}, in which $T_T \simeq T_{mk(max)} \approx 200$ ns.

The results of calculating formulas (2) and (3) are shown in Table 2.

We will examine hardware, time, energy and monetary costs in the program/hardware version of $\log_b X$ function execution in the first version, implemented according to the block diagram illustrated.

(6) Разрядность операндов m	(1) Количество БИС для вычисления функции $\log_b x$									(2) Аппаратурные затраты в корпусах		(3) Временные затраты			(4) Энергетические затраты	(5) Стоимостные затраты
	(7) ЦП К1804 ВС1	(8) СУПК1804 ВР1	(9) БМУ К1804 ВУ1	(10) УСА К1804 ВУ3	(11) ПЗУ К556 РТ5	(12) Регистр К155ИР1	(13) Мультиплексор К155 КП12	(14) Счетчик К155 ИЕ7	(15) Логика	(16) Общее число корпусов	(17) Число стандартных плат (240×280 мм²)	(18) Число микрокоманд	(19) Число тактов	(20) Время вычисления T, мкс	P, вт	Стоимость комплектующих БИС, руб. (22)
4	1	–	2	1	7	1	3	2	4	21	0,5	22	58	11,6	7,35	260
8	2	1	2	1	7	1	3	2	4	23	0,5	26	110	22,0	8,4	405
12	3	1	2	1	8	2	3	2	4	26	0,5	30	166	33,2	9,5	560
16	4	1	2	1	8	2	3	2	4	27	1	34	184	36,8	10,0	600
24	6	1	2	1	9	3	3	4	4	33	1	42	318	63,6	12,3	645
32	8	1	2	1	10	4	3	4	4	37	1	50	422	84,4	14,2	760
64	16	2	2	1	14	8	3	4	4	54	1	82	838	167,6	21,8	1225

Table 2

1 Number of Large Scale Integrated Circuits (LSD) to compute $\log_b X$ function
2 Hardware requirements (packages)
3 Time requirements
4 Energy requirements
5 Cost
6 Operand word length m
7 K1804 VS1 CPU
8 K1804 VR1 ACN
9 K1804 VU1 MCU

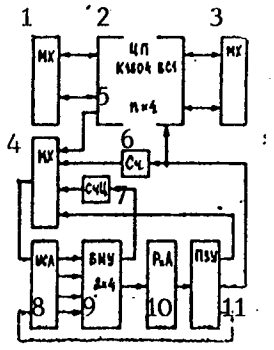
[continued on following page]

[Table 2 continued]

10 K1804 VU3 CN
11 K556 RT5 ROM
12 K155IR1 Register
13 K155 KP12 Multiplexer
14 K155 IE7 Counter
15 Logic Unit
16 Total No. of packages
17 No. of standard PCBs (240 x 280 mm^2)
18 No. of microcommands
19 No. of steps
20 Execution time T, msec
21 R, watts
22 Cost of LSI furnished, rubles

The basic components of a series K1804 microprocessor required for a specialized calculation unit are the four-bit K1804 VS1 central processing unit (CPU), K1804 VR1 accelerated carry unit (ACU), four-bit K1804 VU1 microprogram control unit (MCU), K1804 VU3 MCU control network (CN), and standard integrated circuits such as a series K556 RT5 ROM, K155 IR1 registers, K155 KP12 shift multiplexors, K155 IE7 counters and logic units. Table 2 shows the number of these components and basic cost parameters (as a function of operand length m).

Table 3 shows the corresponding values for the second design variant based on a typical array processing device using nonswitching logic array hardware [4]. This assumes an array processing device (APD) consisting of two autonomous arrays, the control array (COA) and calculation array (CA). The hardware requirements for such a configuration (HR COA and HR CA) are defined in terms of NLA base cells (BC).



1 MKh [not further identified]
2 K1804 VS1 CPU
3 MKh [not further identified]
4 MKh [not further identified]
5 M x 4
6 Counter
7 Dial Counter
8 MCU CN
9 MCU 2 x 4
10 RA [not further identified]
11 ROM

Logarithmic processor block diagram

Таблица 3

(4) Разрядность операндов m	(1) Управляющая матрица: (5) $C_{ум}$, БЯ	(6) Число кристаллов (15) вида 1	(16) вида 2	(2) Вычисляющая матрица: (7) $C_{вм}$, БЯ	(8) Число кристаллов (17) вида 1	(18) вида 2	(3) МВУ для вычисления $\log_B X$: (9) $C_{мву}$, БЯ	(10) Число кристаллов (19) вида 1	(20) вида 2	(11) T, мкс	(12) $P_{потр}$, Вт	(13) Число плат	(14) Стоимость, руб.
4	78	1	—	32	1	—	110	1	—	0,49	0,28	—	35
8	358	—	1	224	1	—	582	1	1	2,49	1,0	—	70
12	922	—	2	576	1	1	1498	1	3	4,7	4,34	—	140
16	1742	1	3	1088	1	2	2830	—	6	8,4	8,06	—	210
24	4287	—	9	2626	—	6	6913	—	15	18,8	18,96	0,5	450
32	7581	—	16	4746	—	10	10481	—	26	38,3	38,42	0,5	910

Table 3

1 Control array
2 Calculation array
3 APD to compute $\log_b X$
4 Operand length m
5 HR (BC)
6 No. of chips
7 HR (BC)
8 No. of chips
9 HR (BC)
10 No. of chips
11 T, microsec.
12 R cons., watts
13 No. of boards
14 Cost, rubles
15 Type 1
16 Type 2
17 Type 1
18 Type 2
19 Type 1
20 Type 2

A comparison of the results in Tables 2 and 3 shows that MPDs based on NLAs have advantages in terms of speed, power consumption and compact size regardless of the operand length involved and, at word lengths of $m \leq 24$, they are superior form the cost point of view.

Proceeding from the uniformity of the recurrent correlations used to calculate various elementary functions by means of the "digit by digit" method, it follows that the structures (configurations) of these computers, either based on microprocessors or on NLAs, are invariable and that they can be used for other functions. In this, both the first and second versions have equal hardware compatibility factors [5]. The advantages of microprocessor-based function expanders are seen only with algorithms involving more complicated tasks, such as the solution of linear and differential equations systems, filter applications and spectral analysis.

The time involved in designing and producing NLA-based array processing devices is comparable to that of similar devices based on microprocessors.

BIBLIOGRAPHY

1. Smolov, V. B., Shumilov, L. A. "Mikroprotsessory i mikro-EVM" [Microprocessors and Microcomputers]--Leningrad: LETI, 1981. 86 pp

2. "Proyektirovaniye spetsializirovannykh BIS na baze bipolyarnykh NLM" [The Design of Specialized Large-Scale Integrated Circuits Based on Bipolar Nonswitching Logic Arrays]. Krys', A. I., Meshcheryakov, V. M., Shumilov, L. A.--Dep. rukopis', 1982, No. 2 (124), No. 47. 54 pp

3. Baykov, V. D.; Smolov, V. B. "Apparaturnaya realizatsiya elementarnykh funktsiy v TsVM" [Hardware for the Calculation of Elementary Functions in Digital Computers]. Leningrad: LGU, 1975. 96 pp

4. "Postroyeniye spetsializirovannyx BIS dlya vychisleniya elementarnykh funktsiy sin Q, cos Q na baze bipolyarnykh NLM" [Design of Specialized Large-Scale Integrated Circuits for the Processing of Basic sine Q, cos Q Functions Using Bipolar Nonswitching Logic Arrays]. Andraus Sueydan, Krys', A. I. - Dep. rukopis', 1982, No. 1 (124), No. 53. 20 pp

5. Alkhovik, A. S., Baykov, V. D. "Mnogofunktsional'nyye matrichnyye
5 konveyyernye vychislitel'nye ystroystva" [Multifunction Array-Conveyance Computing Devices]. Izv. vuzov SSSR--Priborostroyeniye, 1982, No. 4. pp 48-52

The paper was recommended by the Department of Computing Engineering.

12746
CSO: 1863/174

UDC 681.325.2+681.327.68:778.38

EXECUTION OF CELLULAR LOGIC OPERATIONS IN SPATIALLY CONTINUOUS, BIT SLICE PROCESSOR

Leningrad IZVESTIYA VYSSHIKH UCHEBNYKH ZAVEDENIY: PRIBOROSTROYENIYE in Russian Vol 27, No 5, May 84 (manuscript received 20 Oct 83) pp 53-56

[Article by V. M. Denisov, Yu. N. Matveyev, Ye. F. Ochin, Leningrad Institute of Precision Mechanics, and Optics]

[Text] The representation of multilevel images by spatially continuous bit slices is introduced. The algorithm for executing cellular logic operations in the case of such images is examined.

Cellular logic is widely used in order to develop algorithms for the digital processing of images [1,2]. A series of processing structures using cellular logic operations has been developed. The chief drawbacks of such structures are either large time requirements for processing high-resolution images (those with a large number of sample points in the digitized image matrix) when these images are scanned with small array processing units, or large hardware requirements in forming large array processing units (currently there are processors with arrays consisting of 128 x 128 elements). The present article proposes an algorithm for executing cellular logic operations in order to represent images by spatially continuous bit slices, which permits the processing of higher resolution images using only one processing unit [3].

In any physically implementable system a two-dimensional signal $A_0(x,y)$, $x, y \in (-\infty, +\infty)$ is limited in space

$$A_1(x, y) = A_0(x, y) \cdot \mathrm{rect}\frac{x}{2x_{\max}} \mathrm{rect}\frac{y}{2y_{\max}},$$

where $(2x_{\max} \times 2y_{\max})$ is the signal analysis field.

In some analog processes, the signal A_1 is digitized,

$$A_2(x, y) = \sum_m \sum_n A_1(m\Delta x, n\Delta y)\delta(x-m\Delta x, y-n\Delta y),$$

where Δx, Δy are digitization steps; $(m,n)\in M \times N$; $\delta(x,y)$ is the Dirac delta function,

In a digital processor the A_1 signal is not only digitized but also quantized

$$A_3(x, y) = \sum_m \sum_n \sum_i a_i(m\Delta x, n\Delta y)2^i\delta(x-m\Delta x, y-n\Delta y),$$

where $i\in[0,1-1]$, $a_i\in\{0,1\}$, I is the number of digits in the binary representation A_1 $(m\ \Delta\ x,\ n\ \Delta\ y)$.

The A_1 signal can also be presented in a quantized spatially continuous form (QSC form):

$$A_4(x, y) = \sum_{i=0}^{I-1} a_i(x, y)2^i, \qquad (1)$$

where $\{a_i(x,y)\}_{i=0}^{I-1}$ is the set of spatially continuous bit slices (SCBS) for signal A_1.

Representation (1) contradicts the principles of digital data processing and does not agree with known analog optical image processing principles. The overlapping of analog (spatially continuous) and digital (quantized by level) processing opens the way to the development of processes which offer several advantages over the analog and digital processes of the traditional structure. The basic implementation of such analog-digital processes is found in the spatially continuous logic unit (SCLU) which executes the following set of spatial logic micro-operations:

$$\left.\begin{aligned} a(x, y) &= \overline{b(x, y) \vee c(x, y)} \\ a(x, y) &= b(x - \Delta x, y - \Delta y) \end{aligned}\right\}, \qquad (2)$$

where $a(x,y)$ $b(x,y)$, $c(x,y)\in\{o,1\}$, $v(x,y)\in(2x_{max}, 2y_{max})$; Δx, Δy are quantities of spatially shifted functions.

The function arguments (2) are either the results of spatially continuous analog-digital transformations or intermediate operands stored in memory (e.g. holographic).

Many operations employed in image processing are linear in nature. Discrete linear operations used in two-dimensional processing can be described by means of a generalized linear operator [1]:

$$P(k, l) = \sum_{m=1}^{M} \sum_{n=1}^{N} F(m, n) H(m, n; k, l), \qquad (3)$$

where F(m,n) is an array of MxN elements representing the initial (input) image; P(k,1) is an array of KxL elements describing the transformed (output) image; H(m,n; k,1) is the operator core which makes up the set of weighting factors.

We will consider the implementation of a class of operators (3) on the basis of functions (2) using the example of cellular logic operations (CLO) [2]. In developing algorithms we will concentrate on the idealized computer structure shown in Fig. 1. H operator core weighting factors are stored in the microcomputer, images are stored in the spatially continuous bit slice unit memory in QSC form and the SCLU executes the function system (2).

Expression (3) is written as follows:

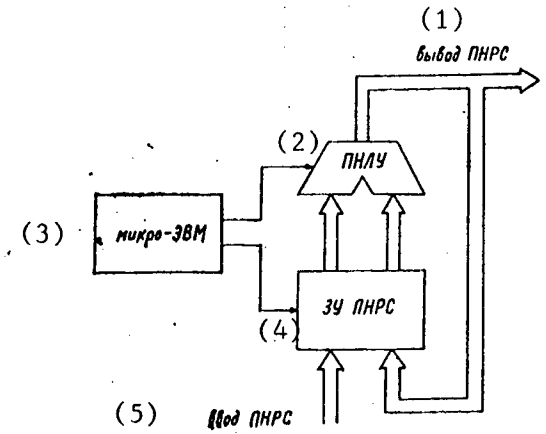
$$P(x, y) = \sum_{m, n \in R^r} T_{m, n}[A_1] H(m, n), \; x, y \in (-\infty, +\infty), \qquad (4)$$

where R^r-r is the point vicinity of the (x,y) point of image A_1; $T_{m,n}[A_1]$= $A_1(x-m\Delta x, y-n\Delta y)$ is the shift operator; Δx, Δy are the minimum shift quantities defined by yo [lists of Russian abbreviations give yo as "limiting amplifier" (usилитel'-ogranichitel'); once yo is given as accelerator-amplifier (uskoritel'-ogranichitel')--Editor] characteristics.

As a rule, cellular logic operations take place in the vicinity of an R4 Neimann or an R8 Moore background (Fig. 2,a and 2,b, respectively). Cellular logic operations (4) in the Moore vicinity have a quantified spatially continuous form of the type

$$P(x, y) = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} 2^{i+j} \sum_{m=-1}^{1} \sum_{n=-1}^{1} a_i(x-m\Delta x, y-n\Delta y) h_j(m, n), \qquad (5)$$

where I,J is the amount of digits in binary representations A_1 and N, respectively; $a_i(x,y)$ is the i-th spatially continuous bit slice of image $A_i(x,y)$; $h_j(m,n)$ is the j-th bit slice of H(m,n).



1 SCBS output
2 SCLU
4 SCBS memory
3 Microcomputer
5 SCBS input

Fig. 1. SCBS processor structure

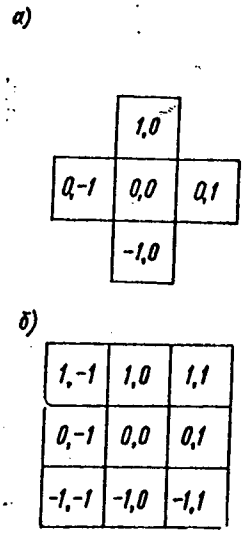


Fig. 2. Niemann (a) and Moore (b) background vicinity configuration

The structure of the microprogram to compute expression (4) through micro-operations (2) takes the following form ($R_k(x,y)$ is the k-Θ SCBS result P(x,y), Kε[0,K-1):

```
for k in ∅ ... I + J + 3 loop
    P_k(x, y) = ∅;
end loop;
for i in ∅ ... I − 1 loop
    for j in ∅ ... J − 1 loop
        for m in −1 ... 1 loop
            for n in −1 ... 1 loop
                if h_j(m, n) = 1
                    then do b(x, y) = a_i(x − mΔx, y − nΔy);
                        for k in i + j ... i + J + 3 while b(x, y) ≠ ∅ loop
                            b'(x, y) = b̄(x, y);
                            P'(x, y) = P̄_{i+J+k}(x, y);
                            Π(x, y) = P̄'(x, y) ∨ b'(x, y);
                            P'(x, y) = P̄_{i+J+k}(x, y) ∨ b(x, y);
                            P_{i+J+k}(x, y) = P̄'(x, y);
                            b(x, y) = Π(x, y);
                        end loop;
                end if;
end loops i, j, m, n
```

Instructions with the body of the loop are executed in the SCLU while the remaining instructions are executed in the [control unit (ustroystvo upravleniya= control, control device, control unit, monitor)]. K=I+J+]$\log_2$(r+1)[=I+J=4 is the length of the result; b(x,y), b'(x,y), P'(x,y) = SCBS for storage of intermediate results; Pi (x,y) = SCBS carry.

Analysis of the microprogram shows that the CLO processing time is $TIJrn_o t_{ts}$ where n_o is the mean number of SCLU work cycles needed to process one image slice, t_{ts} is the SCLU cycle time. For the vicinity of R8, I=J=8, t_{ts}=3 microseconds we obtain r=9, n_o=10 and T≈17 ms.

BIBLIOGRAPHY

1. Prett, U. "Tsifrovaya obrabotka izobrazheniy" [Digital Image Processing]: Translated from English. Moscow: Mir, 1982. Book 1-312 pp; Book 2-480 pp.

2. Preston, K. et al. "Osnovy kletochnoy logiki s prilozheniyami k obrabotke izobrazheniy v meditsine" [Basics of Cellular Logic with Application to Image Processing in Medicine]. TIIER, 1979, vol. 67, No. 5, pp 149-185.

3. Denisov, V. M., Matveyev, Yu. N., Ochin, Ye. F. "Struktura tsifrovogo optoelektronnogo protsessora mnogourovnevykh izobrazheniy po prostranstvennonepreryvnym razryadnym srezam" [Structure of a Digital Optoelectronic Processor of Multilevel Images by Means of Spatially Continuous Bit Slices]. Theses, reports and communications/IV All-Union School Seminar. Parallel information processing, 4-10 April 1983, L'vov, Session 2, pp. 68-69.

The paper is recommended by the Department of Computing Engineering.

12746
CSO: 1863/174

UDC 681.324

ORGANIZATION OF CONVEYOR PROCESSING OF VECTOR COMMANDS IN MULTIPROCESSOR COMPUTER SYSTEMS WITH A REORGANIZABLE STRUCTURE

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84 (signed to press 5 Sep 83) pp 16-21

[Article by A. A. Zabolotnyy, V. M. Kostelyanskiy, G. M. Lekhnova and D. A. Nedzel'skiy: "Organization of Conveyor Processing of Vector Commands in Multiprocessor Computer Systems With a Reorganizable Structure"]

[Text] Multiprocessor computer systems (MVS) with a reorganizable structure of the PS-3000 type are one of the new trends in the development of MVS. From the point of view of the given operation, the following are the basic features of these MVS:

--availability of vector and scalar commands in the system's architecture,

--availability of m single-type control (scalar) processors,

--wide use of the pipeline principle of processing scalar and vector commands, and

--availability of a common computer resource that is redistributed in time, and which consists of n single-type asynchronously functioning processor components each with microprogram control.

The pipeline organization of processing scalar and vector commands in computer systems was researched in sufficient detail [1, 2]. In [1], the modes of conveyor processing of scalar commands (pipeline processing with interlocks and without interlocks) were examined, mathematical models of a processing subsystem in these modes were developed, and the advisability and effectiveness of implementing a pipeline mode without interlocks wereshown. In [2], a technique was developed for evaluating the processing effectiveness of vector commands in the start-stop (without overlapping) and pipeline (without interlocks) modes. It was shown that use of the pipeline mode of processing vector commands makes it possible to improve the performance of multiprocessor computer systems by 10-30 percent (depending on the type of task that is being performed and the parameters of the multiprocessor computer systems).

In [2], however, effectiveness estimates were obtained without taking into account the hardware costs for implementing the pipeline mode in the assumption that the pipeline of vector commands isn't interrupted. Analytical formulas werecitedhere only for the upper limit of effectiveness estimates of multiprocessor computer systems, although it was shown that the difference between the upper and lower limits was insignificant with large-size request buffers. However, the nature of dependencies for the upper limit of effectiveness estimates doesn't make it possible to analytically evaluate the degree of proximity of estimates to the maximum ones depending on the size of the request buffer.

In the present article, the effectiveness of pipeline processing of vector commands is examined and analytically researched with regard to interrupts of the vector command pipeline and the various dumping disciplines of requests for servicing, as well as the structural and algorithmic aids that make it possible to provide effectiveness estimates that are close to the maximum ones with limited hardware outlays (its cost).

/System Description. Basic Model./ [in boldface] The processing subsystem of the MVS with the PS-3000 reorganizable structure that is being researched consists of m (m = 2, 4) control processors and n single-type processor components (n = 8, 16) that form the computer resource.

The control processor reads the commands from the main memory, deciphers them, generates addresses of the scalar and vector operands, and reads them from the main memory or the registers. When the next pair of scalar operands or like-named components of vector operands (request) is ready for execution, it enters the scalar or vector computer resource respectively (under the condition that it's free). If the computer resource is occupied, then the next request is inserted in the buffer (under the condition that there are free spaces in the buffer). If the buffer is filled, then the control processor is blocked. After generating the last request of the next vector command, the control processor changes to processing the next command.

The computer resource of the processing subsystem executes requests in the order of their readiness. If the request buffer is empty, then the computer resource is idle.

Since all the control processors are identical and they work in parallel and independent of each other on various tasks or branches of a single task, then for evaluating the processing subsystem of an entire multiprocessor computer system it's sufficient to examine the functioning of the basic system, which consists of one control processor, one computer unit that is equivalent in performance to the computer resource and accessible to a single control processor, and a buffer for k requests between them.

We'll present the process of executing the sequence of vector commands in a basic model like this in the form of a two-phase queueing system (SMO) without losses (first phase is the control processor, second phase is the computer device) with a final request buffer between the phases and infinite queueing of vector commands during input of the first phase.

Simulation modelling was used in [2] for determining the lower limit of a processing subsystem's effectiveness (functioning of the basic model was examined with the random replacement of vector commands and the geometrical laws of time distribution for generating and executing vector command requests). In the given study, functioning of the basic model with the random replacement of vector commands and determined times for generating and executing vector commands will be examined for obtaining analytical effectiveness estimates. The correctness of such an assumption is explained by the fact that the random variable, which is the sum l of random variables with the geometrical (exponential ones are for a continuous period of time) laws of distribution, has a negative binomial law of distribution (Erlang distribution of l order is for a continuous period of time), and the larger the value l the greater the probability of time for generating (executing) a vector command that is equal to its mathematical expectation. Since the PS-3000 multiprocessor computer system is oriented towards processing vectors with a dimensionality of vector operands $l \leqslant 256$, the random time substitution of generating or executing a vector command with its mathematical expectation introduces an insignificant degree of error.

We'll use (similar to [2]) the load coefficients of control processor H and computer resource E, which are the ratio of useful operating time of the units to the over-all time of their functioning with regard to downtimes, as effectiveness estimates.

/Reasons That Cause Interrupts of the Vector Command Pipeline,/ [in boldface] In general, the reasons that cause interrupts of the vector command pipeline are the same ones during pipeline processing of scalar commands (informational dependence according to operands and addresses, conditional transfer commands, indirect addressing), however, their influence is substantially different. Thus, during pipeline processing of scalar commands, the indirect addressing of an operand can cause a delay of the pipeline. During pipeline processing of vector commands, the influence of indirect addressing is considerably less as a consequence of the large processing time and the execution of vector commands (typical values of dimensionality are $32 \leqslant l \leqslant 256$).

According to [3-5], conditional transfer commands exert a substantial influence on the effectiveness of pipeline processing of scalar commands.

In the PS-3000 multiprocessor computer system, the introduction of vector commands into the architecture, the organization of "vector" branchings by means of various masking vectors, and hardware implementation for generating masking vectors made it possible to decrease the number of conditional transfer commands and, accordingly, their effect on a subsystem's processing effectiveness. Thus, for example, when implementing some algorithm on the vector operands of dimensionality l with the use of scalar commands, it's necessary l times to execute the ORGANIZATION OF CYCLE conditional transfer command and, when implementing this same algorithm with the use of vector commands, this command is absent (under the condition that the dimensionality of the vector registers is sufficient for storing the vector operands).

Hardware costs for support of the pipeline mode is proportional to the depth of the pipeline (to the number of commands (requests) that are located simultaneously in processing and execution). Since its depth doesn't exceed 8 (according to data [6], its depth is 2-4) for the scalar command pipeline, implementation of the pipeline mode without interlocks doesn't present difficulties. The depth of the vector command pipeline can reach several hundreds (depending on the dimensionality of the vector operands), and implementation of the mode without interlocks will require considerable hardware outlays, as the effectiveness of their use can be insignificant at that time.

For simplifying control of the vector command pipeline and reducing hardware costs, it's advisable to implement the mode with interlocks of the pipeline and with the generation and execution of vector commands according to the order that they follow in the program. In this case, the informational dependence according to operands between the vector commands of processing facilitates (and doesn't hinder as in the case of scalar commands) their pipeline processing. The informational dependence of the WRITE TOMAIN MEMORY vector command on the result of one of the previous commands, the effect of which on the effectiveness of the processing subsystem also will be subsequently investigated, is the basic cause of interrupts (interlocks) of the pipeline for the PS-3000 multiprocessor computer system.

/Evaluating the Effectiveness of a Processing Subsystem During Conveyor Processing of Vector Commands With Interlocks./ [in boldface] The analysis of classes of tasks that are standard for the PS-3000 multiprocessor computer system (such as linear programming with the use of a simplex method, iteration methods for solving integral and differential equations, spectral analysis) shows that their execution times basically are determined by the time of processing and execution of vector commands for the addition and multiplication of numbers with a floating decimal point and the WRITE TO MAIN MEMORY vector command. For example, the basic program cycle of a Fourier rapid transform using base 2 contains 6 addition commands, 4 multiplication commands and 4 commands for writing to main memory. Therefore, we'll examine a technique for evaluating the effectiveness of a processing subsystem in the pipeline mode with interlocks in an example of programs that consist of vector commands of these three types.

We'll designate ω_i $(1 \leqslant i \leqslant 3)$ as the probability of occurrence in the sequence of commands of the i-type vector command; $\mu_1, \mu_2, (\lambda_1, \lambda_2)$ as the rates for generating (executing) requests of the first (second) type of vector command. For vector commands of the first and second type (addition and multiplication), the rate of generating requests doesn't depend on the type of command, but is determined only by the architectural features and parameters of the control processor and the main memory. Therefore we'll assume that $\mu_1 = \mu_2 = \mu$.

In the PS-3000 multiprocessor computer system, the condition $\lambda_1 > \mu > > \lambda_2$ [sic] is performed and the load coefficients are $\rho_1 = \mu/\lambda_1 < 1$, $\rho_2 = = \mu/\lambda_2 > 1$ [sic] i. e. when executing the MULTIPLICATION vector commmand, the control processor is more productive; but when executing the ADDITION command, the computer unit is more productive.

The execution time of the WRITE vector command is determined only by the parameters of the control processor and by the type of interface between it and the main memory. Let's examine the multiprocessor computer system with two types of interfaces. The first type of interface has two complete sets of data buses for sending information from the control processor to the main memory (one complete set of data buses is for sending the address and the other is for data). The second type of interface has one. If the second type of interface is implemented in the multiprocessor computer system, then the execution time of the WRITE command also will be two times greater (for example, when writing 32-bit data). In the PS-3000 multiprocessor computer system, $\rho_3=1$ is correct for the first type of interface and $\rho_3=2$ for the second type. The ratio of rates for generating requests of the main memory to the control processor during the processing of vector commands of the first (second) type and the WRITE command is understood under ρ_3 .

We'll examine the conduct of the two-phase queueing system, which describes the process of executing the sequence of vector commands in the basic model of a processing subsystem, at the moments of departure from it of the next vector command.

During the time of execution by the computer device of the first type of vector command $(\lambda_1>\mu)$, $B=[l(1-\rho_1)]$ requests will leave the buffer, since l requests will be executed; and the control processor will generate $l\rho_1$ requests (l is the dimensionality of the vector operands, $[l(1-\rho_1)]$ is the closest integral that is greater than $l(1-\rho_1)$). If there are $i < B$ requests in the buffer, then the computer unit will be idle part of the time.

During the time of execution by the computer device of the second type of vector command $(\lambda_2<\mu)$, $A=[l(\rho_2-1)]$ requests will be added to the buffer, since the computer unit will execute l and the control processor will generate $l\rho_2$ $(\rho_2>1)$ requests. If there are $i > k-A$ requests in the buffer, then the control processor will be idle part of the time.

If the next command is the WRITE vector command, then the computer unit anticipates its execution by the control processor. Following execution of the WRITE command, the request buffer is released.

We'll examine the functioning of the basic model of a processing subsystem when dumping requests for servicing to the computer unit by batches according to n $(1\leqslant n\leqslant l)$ requests. When n = 1, we have the discipline "first arrived--first released" (as a rule, it's precisely this discipline of dumping requests that is used in the majority of models) that provides the maximum (for the prescribed parameters) performance; however, it also requires maximum outlays of equipment for technical implementation, since it's necessary to provide for receiving, following the readiness of the large number k of parallel processes, selecting requests for servicing, and buffering the results of operations and their "names" from the computer unit for supporting the required sequence of their entry into the registers.

When increasing the size of the batch of requests by n times, the number of parallel processes, which it's necessary to follow, decreases; the regularity

of generating a batch of requests, which can be dumped for execution in the order that they follow in the vector operands (requests inside the batch follow as well in the order that is prescribed by the program) increases. In this regard, the necessity for equipment that buffers the results loses its significance with the aim of providing the correct order of their entry, the equipment for following the readiness of the request batches is simplified, and problems of fixing a program's interrupt point and the possibility of its continuation following interrupt from the stop point are easily solved. However, when increasing the size of the request batch, the effectiveness of a processing subsystem will be lower in comparison with the discipline n = 1 and, therefore, when examining the functioning of a processing subsystem, we'll determine the effect of the size of the request batch on evaluations of its effectiveness.

We'll determine the state of the queueing system through the parameter i, which is equal to the number of requests in the buffer at the moments that the next vector command leaves the system. It's necessary to note in this regard that states with $0 < i < |n\rho_1|$, where $[n\rho_1|$ is the closest integer that is greater than $n\rho_1$, will be absent. We'll designate the interrupt probability of a queueing system in state a_i as P_i. The conduct of a queueing system can be described by the Markov aperiodic chain for which a stationary mode exists [7]. The equation system, which connects the interrupt probability of a queueing system in various states, has the following form:

$$
\begin{aligned}
&(1 - \omega_3) P_0 = \omega_3 \sum_{i=1}^{k} P_i;\\
&(1 - \omega_1) P_{[n\rho_1]} = \omega_1 \sum_{i=0}^{B+[n\rho_1]} P_i;\\
&P_i = \omega_1 P_{i+B}, \quad [n\rho_1] < i < n + A;\\
&P_{n+A} = \omega_1 P_{n+A+B} + \omega_2 \sum_{i=[n\rho_1]}^{n} P_i;\\
&P_i = \omega_1 P_{i+B} + \omega_2 P_{i-A}, \quad n + A < i \leqslant k - B;\\
&P_i = \omega_2 P_{i-A}, \quad k - B < i < k;\\
&(1 - \omega_2) P_k = \omega_2 \sum_{i=1}^{A} P_{k-i}.
\end{aligned}
$$

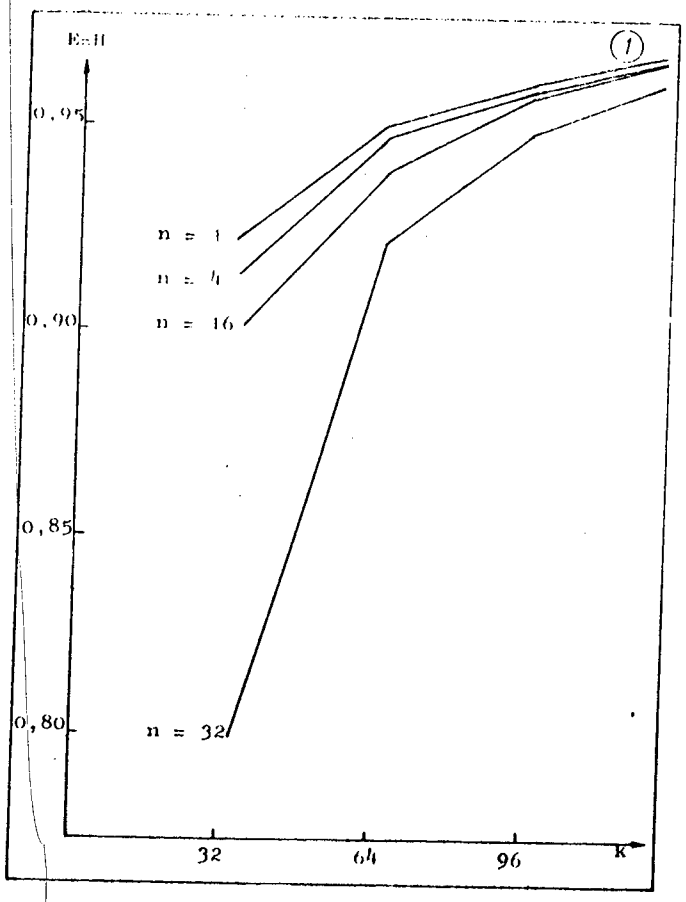


Figure 1. Dependencies of E and H on the Size of the Buffer of Requests k

The determination of load coefficients of the units was performed with regard to the fact that the computer resource is idle part of the time in states $a_i(i<B)$ when executing vector commands if the first type, and the control processor is idle part of the time in states $a_i(k \geqslant i > k-A)$ when generating requests of vector commands of the second type (while executing the WRITE vector commands, the computer resource is idle in all states).

The considerable number of equations hinders the derivation of analytical equations in a clear form for P_i and load coefficients of the units and, therefore, numerical methods were used.

The dependencies of E and H on the size of the buffer of requests k, when $\omega_3=0$ (i. e. in the absence of WRITE commands that cause interrupts of the vector command pipeline), l = 32, ρ_1 = 0.5, ρ_2 = 1.5, and $\omega_1=\omega_2$ = 0.5, are cited in figure 1. As was expected, the maximum values of E and H are attained when n = 1 (i. e. with the "first arrived--first released" discipline). However, while increasing the size of the buffer of requests k, even when n = l ($k=2l$), it's possible to attain practically the same load coefficient values of the units as with n = 1; and, in this regard, equipment outlays for controlling the receipt and dumping of requests and results decrease considerably. An increase in the size of the buffer from l to $2l$ practically doesn't entail an increase in hardware costs with the existing level of integration of memory microcircuits.

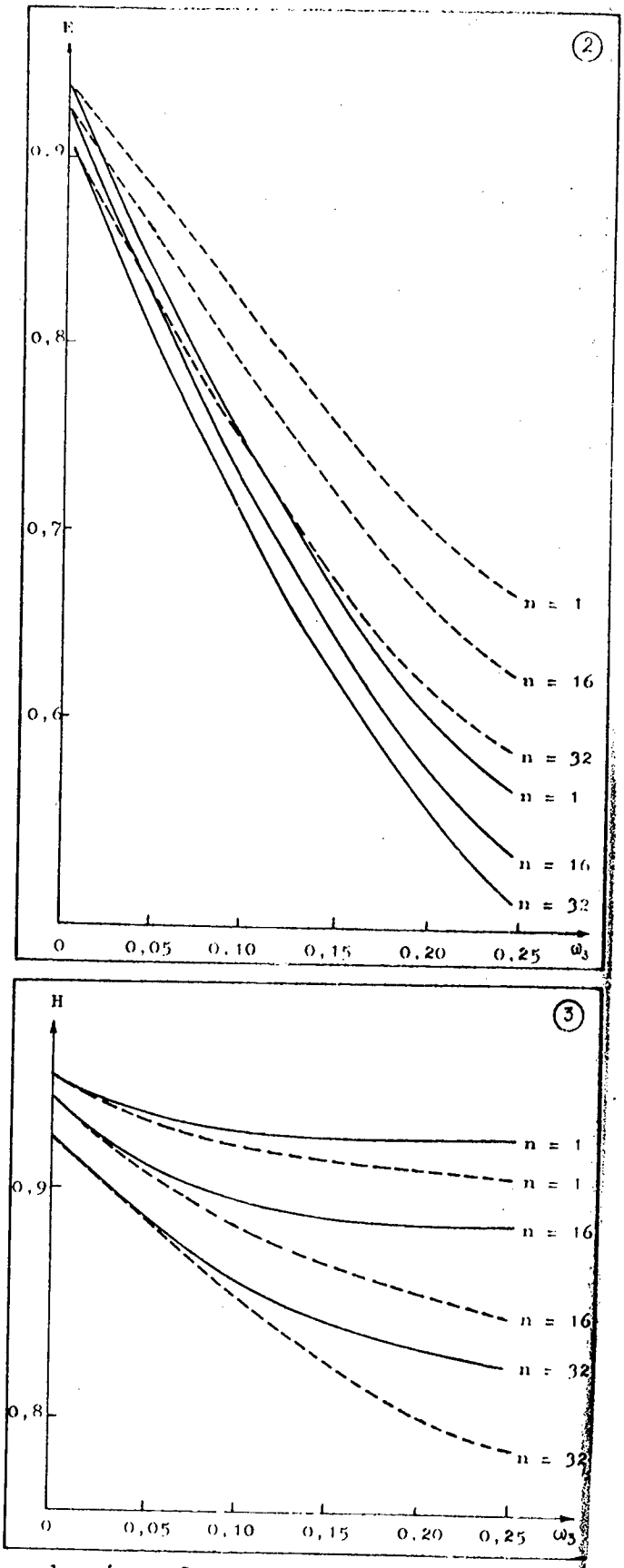
The dependencies of E and H on the interrupt probability of the pipeline of vector commands ω_3 are cited in figures 2 and 3 where k = 64, l = 32, ρ_1 = 0.5, ρ_2 = 1.5, $\omega_1=\omega_2$ = 0.5 ($1-\omega_3$), $\rho_3=1$ is the dashed line, and $\rho_3=2$ is the solid line.

With an increase in ω_3, ρ_3 and the size of the batch of requests n, the load coefficients of the units are decreased, and, moreover, the load coefficient of the computer resource is more substantial. This means that when $\omega_3>0$, the performance of the processing subsystem will be determined by the load coefficient of the computer resource, which is in the range of 0.5-0.7 for typical values of subsystem parameters $\rho_1, \rho_2, \rho_3, \omega_3$. In actual programs--in addition to the ADDITION, MULTIPLICATION, and WRITE commands--there are vector commands, which do not require a computer resource (for example, dispatches to the vector register), as well as scalar commands. Therefore, the load coefficient of a computer resource will even be somewhat lower than the cited values.

Having organized its use with two control processors, it's possible to increase the load coefficient of the computer resource (and then also the performance of the entire processing subsystem). The performance of a processing subsystem with a common computer resource (OVR) for two control processors when there are three types of vector commands with determined generation and execution times, as well as when they are randomly substituted, was examined by a simulation modelling method. The conduct of the queueing system is calculated (as also earlier) at the moments that the next command leaves the computer resource.

As was to be expected, the load coefficient of the common computer resource substantially increased in comparison with the processing subsystem with the individual computer resources (the parameters ρ_1, ρ_2, ρ_3 are identical in the subsystems that are being compared) and when $\omega_3\leq 0{,}15$ is close to one. It's necessary to note that a discipline for dumping requests by batches according to $n=l$ (when using more complex disciplines with $n<l$, the value will be even closer to one), which is a simpler one for technical implementation (and the worst one from the point of view of performance), was examined.

Analysis of the dependence of E on the size of the request buffers shows that buffer $k=2l$ provides for load coefficient values of the common computer resource that are close to the maximum.



Figures 2 and 3. Dependencies of E and H on the Interrupt Probability of the Pipeline of Vector Commands w_3

It was established experimentally that it's possible to use the approximate formula $E=2E_1/(1+E^2_1)$, where E_1 is the load coefficient of the common computer resource with one control processor, for estimating the load coefficient of a common computer resource with two control processors. While yielding to the approximate formula, the degree of error doesn't exceed 3-5 percent in the broad range of altering ρ_1, ρ_2, ω_3.

/Conclusions./ [in boldface] 1. It's advisable to dump requests to the computer resource by batches according to $n \leqslant l$ requests for reducing hardware costs, simplifying control of the vector command pipeline, and providing for the possibility of program continuation when there are interrupts.

2. It's sufficient to have a buffer of requests $k=2l$ in providing for a processing subsystem's performance that is close to the maximum when dumping requests by batches.

3. In the PS-3000 multiprocessor computer system, WRITE vector commands are the basic cause of interrupts of the vector command pipeline that cause a reduction in the processing subsystem's performance.

4. When limiting the cost of a processing subsystem for increasing the load coefficient of a computer resource, it's advisable to make it a common one for two control processors.

BIBLIOGRAPHY

1. Ignatushchenko, V. V., "Conveyor Organization in Controlling Scalar Commands for Multiprocessor Computer Systems," AVTOMATIKA I VYCHISLITEL'NAYA TEKHNIKA, No 2, 1983, pp 57-63.

2. Nedzel'skiy, D. A., "Evaluating the Performance of a Processing Subsystem for Vector Commands of Multiprocessor Computer Systems With a Reorganizable Structure," deposited at VINITI [Scientific and Technical Information Institute], No 6488-82, 14 pages.

3. Flywn, M. J., "Some Computer Organizations and Their Effectiveness," IEEE [Institute of Electrical and Electronics Engineers] TRANS. COMPUT., Vol 21, No 9, 1972, pp 948-960.

4. Ramamoorthy, C. V. and Li, H. F., "Pipeline Architecture," COMPUTER SURVEYS, No 1, 1977, pp 61-101.

5. Brekhov, O. M. and Gutsulyak, Ye. N., "The Effect of Branching Commands on the Performance of Computer Systems," in the book TEORIYA TELETRAFIKA I INFORMATSIONNYYE SETI [Theory of Teletraffic and Information Networks], Moscow, NAUKA, 1977, pp 30-37.

6. Kartsev, M. A. and Brik, V. A., "Computer Systems and Synchronous Arithmetic," Moscow, RADIO I SVYAZ', 1981, 360 pages.

7. Kofman, A. and Kryuon, R., "Queuing, Theory and Application," Moscow, MIR, 1965, 302 pages.

9889
CSO: 1863/69

UDC 681.325.5-181.4

SYSTEMS DESIGN AND APPLICATION OF COMPUTERS WITH FLEXIBLE ARCHITECTURE

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84 (signed to press 23 Dec 83) pp 26-31

[Article by A. V. Palagin, A. F. Kurgayev and A. G. Rokitskiy: "Systems Planning and Application of Computers With Flexible Architecture"]

[Text] For various computer applications, there are different dynamics of formulating the tasks to be solved. In most cases the dynamics are dependent on the intensity of the development of the automation systems, which is connected with the development of the entity which is being automated, the accumulation of knowledge about the entity, and automation experience. Besides, when carrying out experiments in a scientific research laboratory, working out prototypes of a new technique and new technologies, and in other applications, the same computer is often used for the automation of not one, but many entities.

The dynamics of formulating tasks that are being solved demands a certain redundancy of computer components, and, moreover, the introduction of functional redundancy, i.e. the potential capability to alter and expand its functions, is necessary right along with the redundancy of hardware and parameters (memory, complex of external units, speed, reliability). For example, when laying out automation systems, the requirement occurs to jointly use the standard devices of various families of computers. A requirement like this is caused by the fact that the products list and characteristics of both hardware and software aids (applied software in particular) for various families of computers (including within the limits of a single class of computers) are substantially different and supplement each other to a certain extent. In connection with this, the combined use of standard devices from various families of computers improves the quality of the automation systems that are being created or reconfigured, as well as accelerating the process of designing them.

Thus, giving a basic computer the feature of compatibility with the devices of other computers is a highly urgent task.

Provision of the possibility for orienting its internal language (for example, the command system) towards a class of tasks that is being solved and towards

the language of the user is a no less important direction for using the functional redundancy of a basic computer.

Yet another example of the effective use of the functional redundancy of a basic computer is the creation of complexes for modelling automation systems of various entities in real-time. In this case the base computer is used for real-time unit control and for real entities of the control and computing complexes, for the creation of each of which may subsequently be used a specially designed computer or a series computer which is most appropriate for the corresponding entity.

A similar basic computer can also be used as a physical model (tool) for processing basic solutions on the architecture and structure of computers that are newly being developed, for debugging their basic software, as well as a multipurpose computer that emulates the existing models of computers.

A new class of computer--a flexible architecture computer (EVM GA) that is capable of implementing in real time through hardware and microprogramming the architectures of different computers, as well as of being "tuned" to the prescribed internal languages and algorithms--is required as a basic one for all the indicated purposes.

/Computer Architecture./ [in boldface] Definitions of the concept of "computer architecture" that are available in literature are constructive to an insufficient degree from the point of view of setting and solving design tasks. What is in common, which to a greater or lesser degree has been reflected in the definitions of computer architecture which were presented in various publications [1,2], comes down to this assertion: the architecture of a computer is the ordered set of capabilities given to the user.

It's obvious that a computer's internal structure is inseparably connected with the external manifestations of its features, i.e. with the architecture. Maximum support of this unity underlies the successful solution of tasks in design and using computers.

The structure of a modern computer is effectively a composite of programmable automatic devices with a certain hierarchy of control levels that unequivocally correspond to the hierarchy of levels of the architecture: microprogramming, programming (command system), algorithmic (high-level language), and level of operating system and applied programs (systems).

Depending on the characteristics of the tasks that are being solved, the use of an arbitrary aggregate of the computer's architecture levels can be required. In connection with this, it's necessary that each of the levels have at its disposal two functionally universal means:

--presentations (by means of the language) and deliveries (by means of an external interface) of information from external sources, generally signals

(continuous and digital, measurable and adjustable), documents (ordered alphanumeric data), images (visual, acoustical and others) and algorithms, and

--interrelationships (by means of translation and internal interface) with higher levels of the architecture.

However, this doesn't mean that all levels of structure in modern computers are reflected in the architecture, i. e. they are accessible to the user. For example, often the microprogramming level of control performs only the functions of interpreting the upper (programming) level of the operators, while not having a communications interface with the user.

On the basis of the arguments that are cited above, we'll represent the architecture of a computer with the set

$$\Psi = \{\sigma_k\}, \quad k = 1, \ldots, n$$

where σ_k is the complete image of the k level of architecture, which is interpreted in the following manner

$$\bigvee_k \sigma_k = (I_k, J_k, L_k)$$

where I_k is the description of information that is accessible to the k level, J_k is the interface of k level that is represented by a description of its characteristics, and L_k is the language of the k level that is represented by a description of the syntax, semantics and characteristics of its implementation.

In turn, the components (I_k, J_k, L_k) of the k level of architecture are represented by the following sets of its characteristics:

$$I_k = \{I_{k,m}\}, \quad m = 1, \ldots, M_I$$
$$J_k = \{J_{k,m}\}, \quad m = 1, \ldots, M_J$$
$$L_k = \{L_{k,m}\}, \quad m = 1, \ldots, M_L$$

where $I_{k,m}$, $J_{k,m}$ and $L_{k,m}$ are descriptions of the following corresponding characteristics:

$$M_I = \text{Card } I_k; \quad M_J = \text{Card } J_k; \quad M_L = \text{Card } L_k.$$

A list of the basic characteristics of information, interface and language of the programming level of architecture is cited below as an example.

/Characteristics of Information/ [in italics]

1. Information units that are accessible at the programming level (of fixed word length--bit, byte, 16-bit, 24, 32 and others; random word length).

2. Over-all number of data formats.

3. Variable numbers:

--kinds of numbers (whole, rational, complex and others),

--variation range according to kinds of numbers,

--position of the dividing point in a number (before the leftmost digit, after the rightmost digit; floating point),

--basis of the number system,

--accuracy of representation (random, limited; word length of number, order and mantissa), and

--representation of positive and negative numbers (in direct code, additional and reverse).

4. Symbols:

--set of alphabets, and

--coding of symbols.

5. Memory:

--kinds of memory (permanent, semipermanent, long-term, main, register, logic-in--stacked, and associative), and

--characteristics of each kind of memory (physical word length, addressable units of information, variation limits of memory volume, access time, storage reliability, methods for controlling and recovering information, and others).

/Characteristics of the Command System/ [in italics]

1. General characteristics:

--application scales (names of computers that use a given command system; date of first delivery of each type of computer, number of each type of computers that are located at the users; annual volume of output),

--basic areas of computer application with a given command system,

--number of commands,

--capacity of the command system--relative characteristics that evaluate the functional resources of a given command system with respect to a standard command system (it is determined by the ratio of the interpreter's volume of a given command system, which is written in a standard command system, to the

interpreter's volume of a standard command system, which is written in a given command system), and

--expandability of the command system (number of free codes, coding of commands for accessing the expander, characteristics of the expander's command system).

2. Coding of commands:

--word length of commands,

--number of command formats, and

--command formats: word length of fields in the command word, connection of each of the fields with the others, semantic content of the fields and codes.

3. System of operations:

--over-all number of operations,

--arithmetic operations,

--logic operations,

--control (of sequence of command exeuction, of data),

--forwarding and exchange,

--stacking operations,

--with bits,

--input-output,

--generation of effective address, and

--console and systems operations.

4. Addressing:

--number of addresses of commands (zero-, one-, two-address and others),

--number of operands that are addressable in one command,

--maximum addressable memory capacity,

--word length of the effective address,

--number of levels of indirect addressing,

--coding of indirect addressing flag, and

--program-accessible registers: over-all number; number of common registers, index ones, segment registers, A-registers, stacking ones and others; conformity of addresses to functional groups of registers.

5. Interrupt system:

--type (priority, nonpriority, single-level, multilevel),

--number of levels,

--method for establishing priority (programming, microprogramming, hardware),

--sources of interrupts (programs, console, timer, input-output unit, control circuits, and others),

--structure of interrupt register (word length, number of interrupt causes), and

--structure of word of program status (over-all word length, number of fields, their word length and semantic content).

6. Characteristics for implementing a command system in a specific computer:

--speed: time vector $t = (t_1, \ldots, t_i, \ldots, t_n)$ of execution of commands, average time for solving a standard task, and

--reliability: vector of rates of failure in execution of commands (λ_i is characteristics); vector of probabilities for proper execution of commands for a standard mixture of commands $p = (p_1, \ldots, p_i, \ldots, p_n)$, where $p_i = \exp(-\lambda_i \alpha_i t_i)$, α_i is the relative frequency of using the i command, and n is the number of commands; probability for proper execution of a standard mixture of commands $P = \prod_{i=1}^{n} p_i$; and probability for proper solution of a standard task.

/Characteristics of a Computer Interface/ [in italics]

1. Indicators of quality:

--level of standardization (national, international),

--connectedness (it is absent, the source to receiver, the source to n receivers),

--type of communications (radial communications, main, chained),

--load capacity of data buses,

--data transmission speed,

--reaction time to interrupt,

--number of lines (including control, data),

--control of reliability of information that is being transmitted, and

--priority structure (fixed, absolutely or relatively changing).

2. Functional and structural organization:

--structure of data buses (number of lines according to types of unibuses (data, address, control).

/Flexibility of Computer Architecture./ [in boldface] We'll evaluate the flexibility of architecture of a computer that is complete in design with a set of architectures that are being implemented (accessible) in it without altering the structure. In this regard, we'll represent the computer's architecture with the matrix

$$\Psi_A = \| \sigma_k^a \|, \ a = 1, \ldots, A \tag{1}$$

where A is the number of architectures that are being implemented on the basis of a given computer; $\sigma^a{}_k$ is the complete image of the k level of type a architecture.

We'll represent the description of the levels in the following form:

$$\underset{k,a}{\forall} \sigma_k^a = (I_k^a, J_k^a, L_k^a)$$

where $I^a{}_k$, $J^a{}_k$ and $L^a{}_k$ are symbols that correspond to the descriptions of components in connection with type a architecture. A complete description of each of the components can be represented in the form of a matrix. Thus, for example, in connection with the k-th component

$$I_k^a = \| I_m^a \|, \ m = 1, \ldots, M_I .$$

While examining the various computers in accordance with the definitions cited above, it's possible to come to the conclusion that a certain flexibility is inherent in their architectures. To a greater degree, this is manifested at the higher levels of control (systems, algorithmic, operating system) by placing configuration aids, applied program packs, algorithmic languages, operating systems, and others at the disposal of the user. To a lesser degree, it's manifested at the levels of the command and microcommand system of modern computers.

In connection with the class of flexible architecture computers, it's advisable to characterize an architecture's flexibility with the set

$$\Xi = \{\Psi_a\}$$

the elements of which are described in accordance with (1).

A list of types of (computer) architectures or their over-all number that are being implemented can provide a brief description of an architecture's flexibility.

A vector of the following form can serve as well as a brief quantitative description of flexibility

$$G = \{g_I, g_J, g_L\}$$

where g_I, g_J and g_L are the number of components of the proper type (kinds of information, interfaces, languages) with possible itemization according to the levels of k, as for example

$$g_I = \sum_k g_{I_k}.$$

/Programming Compatibility./ [in boldface] As was noted, computer architecture is a hierarchical system of control levels, the functioning of each of which is provided by the levels that are located below. In this respect, a special role is allotted to the command system level that, as a rule, is the lowest out of those that are being made available to a user of modern computers.

Thus, compatibility at the command system level or programming compatibility, which is understood as providing the possibility of executing one and the same program in different computers, is one of the most important problems of compatibility. In this regard, three kinds of compatibility are singled out: compatibility between computers of different manufacturers, compatibility between computers of different generations of the same manufacturer, and compatibility between models (from lower to higher) within the same generation of computers of the same manufacturer.

By the present time, six basic methods of providing for programming compatibility have been defined: manual reprogramming, translation, simulation of one computer on another, metalayout, decompiling and emulation [3]. Out of the enumerated methods of providing for programming compatibility, emulation is the most universal and effective one that uses a combination of hardware and software simulation aids.

/Emulation./ [in boldface] Emulation, like the other methods too of providing for programming compatibility of flexible architecture computers with different computers, solves three basic tasks: providing for informational compatibility, interface compatibility, and operator (command) compatibility.

Solution of the first task consists of coordinating the formats of data and commands in the areas of storage and the transmission and conversion of information. The following are the most important fragments for solving this task: synthesis of the memory structure and determination of the structure of the information, address and control unibus system. Solution of the given task is substantially simplified with the emulation of informationally compatible computers (for example, SM-1 and SM-3 or SM-2 and SM-4). [sic]

Solution of the second task consists of effectively implementing an input-output interface of a flexible architecture computer that provides for the operation of a basic complete set of peripheral units under emulation conditions or the possibility of arranging a flexible architecture computer

with a complete set of external units that are specifically for each of the types of computers that are being emulated.

Solution of the third task consists of developing hardware and microprogramming means that provide for the effective implementation of the command language of each of the computers that is being emulated. The following are the most important fragments for solving this task:

--selection of a sufficient microoperating base and synthesis of microcommand formats,

--development of multipurpose means for analysing codes (for example, coded combinations of different word length with random arrangement in the command word),

--development of a developed system for generating the effective address of operands,

--development of a multipurpose system for addressing microcommands,

--selection of the number and word length of microprogram and program accessible registers,

--development of the structure for a register of process status and means for analysing its contents,

--development of functional expansion means for a microprogramming control system, and

--development of means for the interaction of architectures, for example, of programming transition from one command system to another.

The quality of solving the indicated tasks determines the effectiveness of a flexible architecture computer under the conditions of emulation. The effectiveness of a flexible architecture computer can be substantially improved by virtue of using a progressive industrial component base and implementing absolute principles for over-all organization of the computer process (pipeline [in English italics], pipeline and parallel data processing, controlled synchronization).

When formulating the command system of new computers in flexible architecture computers, one isn't always successful in full measure in using emulation. Supplying the user in the complex with the flexible architecture computer of a system for automating microprogramming and arranging the computer and control systems on the base of a flexible architecture computer must facilitate this process.

The NCR firm's NCR/32 SBIS [very high-speed integrated circuit], the production of which began in 1983, can serve as an example of technical solutions that are being used when implementing flexible architecture computers. The NCR/32 complete set envisages the possibility of operating with external microprograms

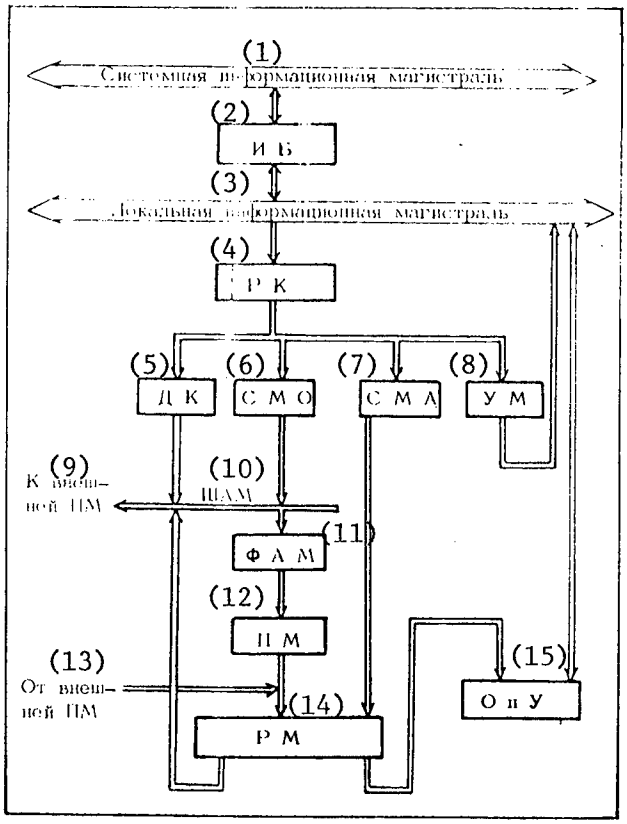


Figure 1. Structural Diagram of a Flexible Architecture Computer

Key:

1. Systems information unibus
2. Information base (IB)
3. Local information unibus
4. Command register (RK)
5. Command decoder (DK)
6. Queueing sytem (SMO)
7. Mass addressing sytem (SMA)
8. Masking junction (UM)
9. To external programming memory (PM)
10. Microcommand address data bus (ShAM)
11. Microcommand address generator (FAM)
12. Microprogramming memory (PM)
13. From external programming memory
14. Microcommand register (RM)
15. Operating unit (OpU)

that are stored in the memory with a capacity up to 128K bytes. On the basis of such a microprogramming system, the NCR/32-000 microprocessor that is included in the complete set makes it possible to effectively emulate the machine language of practically any computer.

The Varian Data Machines firm's models of emulating computers [5], which are distinguished by multipurpose means for decoding command codes of different formats, a flexible input-output system, and aids for functionally expanding the microprogramming control system, can serve as other examples.

The structural diagram of a flexible architecture computer, which illustrates the basic mechanisms for "tuning" the architecture, is described in [6] (figure). Here RK is the command register, DK is the command decoder, UM is the masking junction, FAM is the microcommand address generator, PM is the microprogramming memory, RM is the microcommand register, ShAM is the microcommand address data bus and OpU is the operating unit.

Microprogram controlled selectors--SMO [queueing system] and SMA [mass addressing system] multiplexers--are used for allocating the operation code fields and the number of the general purpose register. Programmable logic arrays, memory with linear selection or an associative memory unit can be used as a command decoder. When the emulation of several command systems is necessary, the command register expands to a certain number of bits for storing the code of the command system's number that is being emulated at the current moment of time. Replacement of the code of the command system's number leads to switching of the command decoder's operating conditions.

/Evaluation of the Effectiveness of a Flexible Architecture Computer./ [in boldface] First of all, we'll note that a flexible architecture computer can be characterized by the flexibility indicators that were examined above.

The integral absolute indicators, which characterize the physical volume, the consummable capacity and cost of a flexible architecture computer, and the performance and reliability of executing programs in it that are written in the languages of the command systems of the computers that are being emulated, are important as well.

Right along with this, the relative indicators, which characterize the effectiveness of decisions that are made when implementing the structure of flexible architecture computers, can represent considerable interest. For example, we'll examine the relative indicator of a flexible architecture computer's performance in comparison with a certain k of the computer that is being emulated.

The solution for k of a computer of certain class H tasks with an average relative frequency vector $\alpha = \{\alpha_i, i = 1, \ldots, N_k\}$ for using this computer's commands can be characterized as time vector $t = \{t_{ik},\ i = 1, \ldots, N_k\}$ for execution of the commands. Then the average time for solving class H tasks is

$$T_k = \sum_{i=1}^{N_k} \alpha_i t_{ik}.$$

In turn, the execution time of each of the commands for any microprogramming computer can be determined by the sum

$$t_{ik} = \sum_{j=1}^{M_k} \beta_{ijk} \cdot t_{jk}, \qquad (2)$$

where M_k is the capacity of the microcommand set of k computer; β_{ijk} is the absolute frequency of using j microcommand when executing i command of k computer.

However, the execution time of different microcommands is equal to or divisible by a certain constant t_k that is called a microcommand cycle and selected proceeding from the time characteristics of the components that are being so that $t_{jk} = a_{jk} \times t_k$ ($a_{jk} = 1, 2, \ldots$).

Then formula (2) can be represented in the form

$$t_{ik} = \sum_{j=1}^{M_k} \beta_{ijk} \cdot a_{jk} \cdot t_k = t_k \sum_{j=1}^{M_k} \beta_{ijk} \cdot a_{jk},$$

and for T_k we'll finally obtain

$$T_k = t_k \sum_{i=1}^{N_k} \alpha_i \sum_{j=1}^{M_k} \beta_{ijk} \cdot a_{jk}.$$

When solving this class of H tasks by placing programs of k computer in a flexible architecture computer, we'll obtain a similar formula:

$$T_0 = t_0 \sum_{i=1}^{N_k} \alpha_i \sum_{j=1}^{M_0} \beta_{ij0} \cdot a_{j0},$$

where $t_0, M_0, \beta_{ij0}, a_{j0}$ are the flexible architecture computer's parameters that are similar to the corresponding parameters of k computer.

The ratio T_k/T_o is an effectiveness indicator that characterizes acceleration (retardation) in executing the programs of k computer in a flexible architecture computer when solving class H tasks. Its sensitivity to the time characteristics of components that are being used (t_k and t_o respectively) is a particular shortcoming of this indicator. The indicator is free from this shortcoming

$$\gamma_{ik} = \frac{T_k}{T_0} \cdot \frac{t_0}{t_k} = \sum_{i=1}^{N_k} \alpha_i \sum_{j=1}^{M_k} \beta_{ijk} a_{jk} \Big/ \sum_{i=1}^{N_k} \alpha_i \sum_{j=1}^{M_0} \beta_{ij0} a_{j0},$$

which evaluates the effectiveness of structural solutions that are implemented in a flexible architecture computer in comparison with one of the computers that is being emulated. A similar indicator can be determined in the set of

computers that are being emulated:

$$\gamma_{cpt} = \frac{1}{K}\sum_{k=1}^{K}\gamma_{tk},$$

where K is the number of computers that are being emulated.

/Conclusion./ [in boldface] The modern stage of developing computer technology by right can be called a microprocessor one. The mass assimilation of microprocessing technology means into the most diverse areas of the national economy advanced to the foreground a whole series of problems, the most important of which are the standardization and unification of both the hardware and software of microprocessors, microcomputers and microprocessor systems.

The improvement of the industrial component base of computer technology, the development of microprogramming control principles, and the over-all evolution of computer architecture and structure predetermined the appearance of a new class of computers--flexible architecture computers. Right along with the other possible spheres of application, flexible architecture computers in a large measure are capable of assisting in the solution of the urgent problems that are named above. In particular, a way of creating standardized means of microprocessor technology, while maintaining and effectively using the expensive stores of software of various models and types of small computers that have already been accumulated, and with the possibility of arranging it on the basis of a basic flexible architecture computer, seems to be advisable and economically advantageous.

In fact, the given article is a brief introduction to the bases of systems programming and the application of flexible architecture computers. The definite practical experience, which was obtained in the given area at the Ukrainian SSR Academy of Sciences Cybernetics Institute imeni V. M. Glushkov, attests to the promising future of the given class of computers (particularly in the "micro" division) and at the same time to the existence of a set of highly complex, but interesting theoretical and applied tasks of general computer theory that for the time being await its solution.

BIBLIOGRAPHY

1. Tanenbaum, E., "Multilevel Organization of Computers," Moscow, MIR, 1979, 547 pages.

2. Kartsev, M. A., "Architecture of Digital Computers," Moscow, NAUKA, 1978, 296 pages.

3. Malinovskiy, B. N. and Pogorelyy, S. D., "Emulation Methods of Computers," UPRAVLYAYUSHCHIYE SISTEMY I MASHINY, No 4, 1974, pp 43-45.

4. "Set of 32-Bit Instruments With a Large Programming Memory," ELEKTRONIKA, No 18, 1982, pp 3-5.

5. Agrawala, A. K., "Foundation of Microprogramming. Architecture Software and Applications," New York, Academic Press, 1977, 420 pages.

6. Palagin, A. V. and Rokitskiy, A. G., "Flexible Architecture Microcomputers," MEKHANIZATSIYA I AVTOMATIZATSIYA UPRAVLENIYA, No 3, 1983, pp 10-14.

9889
CSO: 1863/69

UDC 681.3.65

PROBLEMS IN ORGANIZATION OF DYNAMIC MICROPROGRAMMING

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84
(manuscript received 9 Jan 84) pp 52-57

VASENDO, V. G.

[Abstract] Problems in organization of dynamic microprogramming are analogous to those in the case of programming. The basal problems in the case of dynamic microprogramming are storage of large microprogram arrays and assurance of minimum access time. This requires an appropriate storage layout, typical examples being the fast-access M-storage in B1700 hierarchical structures and the read-write control storage disks in 1600/30 Microdata computers. A second problem here is the algorithm of loading microprograms into the storage, which can be done without or with program control. The latter method is preferable, although slower, because the loading operation becomes independent of the control processes and can proceed simultaneously with other operations as well as during computer operation, also because it is simple and requires no special instruction codes in addition to standard ones. A third problem here is transfer of control to special microprograms, for which there are three methods available and any two of them can be combined. Transfer can be effected by arbitrary interpretation, by fetch instructions, and by means of inactive or only sporadically otherwise used instruction codes, CAL DATA 135 being an example of combining the first two methods. The system problems of dynamic microprogramming relate essentially to multimicroprogramming with optimum allocation of resources, using the concept of "virtual microprogram storage" so as to ensure interchangeability and synchronization of microprograms as well as protection of basic microprograms against interference from special ones and protection of special ones in one array from those in another. A second system problem is integration of software with the operating system, especially for expansion of user categories. This is done on the conceptual level so as to retain the basic system characteristics while realizing all functional capabilities without loss of error immunity, and on the structural level so as to include all microprogram components in the structure of control programs and ensure proper interaction and matching of modules. A third system problem is accessibility from the user's standpoint, most expediently ensured by automation of terminal facilities but also requiring laborious coding of microprograms with necessary verification, simulation tests, and hardware debugging. A separate problem

is mobility of microprograms, inconsistent with mobility of programs within a computer family. Possible solutions to this problem can be construction of computer-independent microprogramming languages and translators, standardization of microinstructions, or duplication of microprogrammatically realized portions of programs with automatic adjusting of programs for either programmatic or microprogrammatic execution. References 16: 4 Russian, 12 Western (1 in Russian translation).
[68-2415]

SOFTWARE

STREAMLINING THE RUSSIAN SOFTWARE INDUSTRY

Moscow IZVESTIYA in Russian 17 Oct 84 p 2

[Article by V. Kipayev, professor, distinguished scientist of the RSFSR: "Programming Industry"]

[Text] High program quality is an important prerequisite for the effective introduction of computers in the economy.

Without a program the computer is nothing but an "empty brain", an expensive but useless apparatus. Thus a new problem has arisen: the organization of the mass industrial production of computer programs on the basis of modern technology.

The economic potential of nations in today's conditions is largely determined by the level of production technology of various industrial products. Technology determines to a significant degree the labor productivity, production quality and prospective production growth. Control and data processing programs have become industrial engineering products. And they need the high efficiency technology which permits large, high-quality programs to be produced under strict time constraints with minimal labor input.

The transition from manual production of products to their industrial production is quite characteristic of the process of development of various areas of human activity. Here, a large collective of specialists of various qualifications replaces the individual craftsman. Each specialist is responsible for his own stage of the total process of the final product's manufacture.

What has been said is largely applicable to the creation of computer programs as well. The industrial technology for their design began to develop relatively recently, and still has not been completely formulated as an independent field of industry and applied science. Recently, however, the interest of specialists in various branches of the economy in program design methods has increased sharply, since time expenditures entailed in the creation of many systems utilizing computers have come to be dictated by the time required for the preparation of their programs.

The manual methods which still exist in certain organizations, by means of which control programs are written (for electric power stations, robots, manufacturing processes, flexible automated production facilities), make it necessary to spend 5-7 years to write a 100-200 thousand line program. Meanshile, the control system hardware is designed and installed in a period of 3-4 years. It need hardly be pointed out what the result of this discrepancy in the development period of the "electronic brains" themselves and their mathematical contents will be. It is precisely here that one of the primary causes for delays in the implementation of various automated control systems in the economy lies.

It should be kept in mind that the cost of the programs comprises more than one half of all expenditures for the computer system in control systems for integrated objects and industrial processes. And this is fully justified: the cost of writing a single instruction in complicated program sets comprise 20-50 rubles, and more than 100 rubles in certain instances. And there are tens and hundreds of thousands of such instructions in the program set.

Programs are not only expensive. They are very complex and thus require careful checking and thorough testing. An error in a control program for robots or flexible automated production facility may be fraught with malfunctions and damage to expensive equipment. An excellent example of the value of program quality control was cited in the letter of Ye. Yakovlyev "The Intention" (IZVESTIYA, No 141, 1983). There it was described how the inclusion of only a single incorrect instruction in the program stopped the primary VAZ assembly line for several hours. Program errors are accompanied in aviation and space flight with a risk to the health and safety of human beings. These circumstances have increased to a significant degree the importance of determining the quality of computer programs.

Expenditures in production technology comprise 15-30 per cent of total production costs for a product in the machine building industry. The role of production technology in electronics and computer fabrication is even greater. However, production technology expenditures in the area of industrial program design are insignificant (2-3 per cent). This underlines the low level of sophistication of their production technology.

The necessity of creating integrated technological systems for the automation of all stages in the creation and operation of control computer complexes has become serious. The introduction of robots at the "Moskvich" automobile factory may serve as an example, where they are supposed to number five hundred and fifty by the start of the coming five year plan. Even if simple control programs of 1-2 thousand line length are utilized in them, then in this instance the total length of the programs comprises millions of lines. A strictly regulated industrial technology and special subdivision are needed for the development and alteration of these programs in accordance with production requirements in an established time framework.

Hundreds of organizations and various departments are currently occupied with the development of programs. However, this work is not planned on a national

scale or coordinated as a whole. The national economic plan and the plans of departments and enterprises do not so much as even mention this type of product.

It is obvious, in addition, that departments and individuals must be designated who are personally responsible for the national organization of the industrial production of various classes of programs. Specialized enterprises and head organizations coordinating their work are necessary for the creation and introduction of program production methods.

Modern software engineering is similar to the computer assisted design of any industrial product. A number of organizations in the nation have already accumulated experience in the effective utilization of powerful universal computers for solving the discussed tasks. The design, duplication, documentation and development of programs have been nearly completely automated.

This enables control system hardware and software to be developed in parallel and thus to significantly reduce the time required for their creation. However, these methods and the facilities required for the automation are as yet not used on a mass scale in this nation. Another circumstance should be considered. Many program sets utilize components with identical functions. Hundreds of practically identical programs can be found today, on which specialists of different departments independently worked, wasting an enormous amount of labor. But surely it is well known that in all branches of industry the delivery of standard assemblies is widespread, which yield large savings. The repeated utilization of worked-out components of programs in the creation of programs can yield a similar savings (it is entirely possible to see them as standard parts). But, of course, standards for the formulation and testing of program modules, and also for their quality, must first be introduced to solve this task.

This standardization permits program development duplication to be liquidated, and to accumulate a high quality program product, which can be repeatedly and widely utilized in systems having different purposes, independently of their departmental affiliation. The implementation of modern methods and facilities that automate them, and of standards in program production will increase labor productivity appreciably, and will reduce significantly the time required for the creation of complex programs and of entire systems. Of no less importance here is the assurance of the high quality of the programs.

The creation of a programming industry entails many problems. The combined efforts of the leaders and manufacturing engineers, programmers and scientists are needed for their solution. The organizational problems involved in the assimilation of this present "no man's land" of industrial production are now especially acute. It is necessary to organize the training of specialists, and publish scientific and industrial literature. An integrated program of research and the creation of fundamental collectives is necessary for the sharp increase in the production volumes and for improving the quality of programs. The manual production of programs must be replaced by modern industrial production. Without a modern industrial production method for computer programs, the continued, effective introduction of the computer into the economy is not possible.

12678
CSO: 8144/0405

UDC 681.3.06

GRAFIT LANGUAGE FOR SIMULATING TWO-DIMENSIONAL GEOMETRIC ENTITIES AND SKETCHES

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84 (signed to press 29 Jul 83) pp 79-80

[Article by V. G. Sirotin: "GRAFIT Language for Simulating Two-Dimensional Geometric Entities and Sketches"]

[Text] /Introduction./ [in boldface] The GRAFIT language [1] is designed for use in computer-aided systems that are oriented towards design and industrial preparation in machine building.

Models of two-dimensional (2D) and so-called 2 1/2 dimensional (2 1/2D) geometric entities and sketches are being built by means of the GRAFIT language in a computer's memory. Entities that can be interpreted as two-dimensional--for example, solids of revolution--are 2 1/2D entities.

The constructed models are subsequently sent to some kind of applied program for making calculations and they are stored in the data base for subsequent operation and modification.

The GRAFIT language can be used both by systems and applied programmers (during the development of SAPR [computer-aided design systems] and SATPP [computer-aided industrial production engineering systems]) and directly by designers and nonprogrammers (for describing sketches of components).

Information of the following types is stored in the models that are being built by means of GRAFIT:

--geometric (coordinates and parameters of elements),

--hierarchical,

--attributive (weight, cost, and so forth), and

--information on contacts of elements between themselves (methods for describing them).

Operation with the language is possible in two modes--interactive and batch. The GRAFIT language is a command-type language that includes approximately 100 different commands, GRAFIT is part of the big system of computer graphics, which includes as well the subsystem for operating with 3D entities, the subsystem for providing dialog, and also a specialized SUBD [data base management system]

/Operation With the GRAFIT Language in the Batch and Interactive Modes./ [in boldface] The GRAFIT langauge was implemented like a batch of FORTRAN programs. With respect to the user, the batch comes in the form of a geometric language--expanded FORTRAN. The expansion is made by virtue of introducing a new type of variables--geometric ones--and statements for operating with them. The statements for operating with the geometric variables are formally constructs designs of the FORTRAN language. For example, T = TO (W1, W2) is the point of intersection of elements W1 and W2; CALL YHEB (W) is editing of element W.

The models, which are built both as batch and interactive versions of GRAFIT, are compatible. This makes it possible with the model of one entity to operate alternately in the interactive and batch modes.

The syntax of the statements of the interactive mode language is slightly different from the syntax of the batch statements. Thus, through their own analogs the statements that were cited above have:

TO WL, W2

YHEB W

The names of the newly determined entities are generated automatically and written on the display screen next to the graphics of the entities. There is a "purge" mode for receiving the graphics and during which the names and auxiliary plots are eliminated.

/Example of a Description of a Sketch By Means of GRAFIT PPP [Applied Program Batches]./ [in boldface] The description of a sketch of a gasket, which was performed by means of a GRAFIT PPP subroutine, is cited below:

```
SUBROUTINE TEST

C  SUBROUTINES PX AND PY DETERMINE
C  THE HORIZONTAL AND VERTICAL STRAIGHT LINES.  IN
C  THE GIVEN CASE, THE STRAIGHT LINES PASS THROUGH
C  THE CENTER--POINT (0, 0)
   P1 = PX (0).
   P2 = PY (0).
C  SUBROUTINE SXY DETERMINES THE SEGMENT THROUGH
C  THE COORDINATES OF THE INITIAL AND TERMINAL
C  POINTS
   S3 = SXY (-100., -100., 100., -100.)
```

```
C  A NEW ENTITY IS GENERATED THROUGH A TURN OF THE
C  INITIAL ONE AROUND ITS CENTER BY MEANS OF
C  SUBROUTINE YVC
   S4 = YVC (S3, 90.)
C  SUBROUTINE YOTP SERVES FOR PLOTTING THE
C  ENTITY BY MEANS OF A REFLECTION
C  RELATIVE TO THE STRAIGHT LINE
   S5 = YOTP (S3, P1)
   S6 = YOTP (S4, P2)
C  SUBROUTINE TKAH DETERMINES THE POINT
C  ACCORDING TO ITS COORDINATES
   T7 = TKAH (45., -35.)
   T8 = YOTP (T7, P1)
   T9 = YOTP (T7, P2)
C  SUBROUTINE D3T DETERMINES THE ARC
C  THAT PASSES THROUGH 3 POINTS
   D10 = D3T (T8, T9, T7)
C  SUBROUTINE SHK DETERMINES THE SEGMENT
C  ACCORDING TO THE INITIAL AND TERMINAL POINT
   S11 = SHK (T7, T8)
C  SUBROUTINE RLS DETERMINES THE LINEAR DIMENSION
C  AND RDK THE DIAMETRIC DIMENSION
   RL12 = RLS (S3, 0., 15., 0, 0.)
   RL13 = RLS (S4, 3., 15., 0, 0.)
   RL14 = RLS (S11, 1., 7., 0, 0.)
   RD15 = RDK (D10, 45., 0, 0., 0.)
C  SUBROUTINE BBP ESTABLISHES THE GRAPHIC OUTPUT
C  MODE; IN THE GIVEN CASE IT IS THE
C  PURGE MODE
   CALL BBP(3)
C  SUBROUTINE BB AGAIN OUTPUTS GRAPHICS OF THE
C  SKETCH WITH REGARD TO THE OUTPUT MODE
   CALL BB
C  CANONIC REPRESENTATIONS OF THE ELEMENTS ARE
C  OUTPUTTED BY MEANS OF SUBROUTINE PKM.  IN
C  THE GIVEN CASE IN THE RANGE OF NUMBERS 1-11.
   CALL PKM(1, 11)
   RETURN
   END
```

A rough draft version of the sketch is depicted in figure 1 and their names are written down beside the elements, and the clean version of the sketch is presented in figure 2.

The so-called canonical parameters of geometric elements, which subsequently can be used for some special calculations, often arouse interest. A list of these parameters for the first 11 elements of the sketch are cited in the table. The coordinates here of one of the points at the straight line are for straight lines (X, Y) and F is the inclination of the straight line to OX in the radians; coordinates of the initial point of a segment are for segments (X, Y) and R is its length; coordinates of the center are for arcs (X, Y); F is the

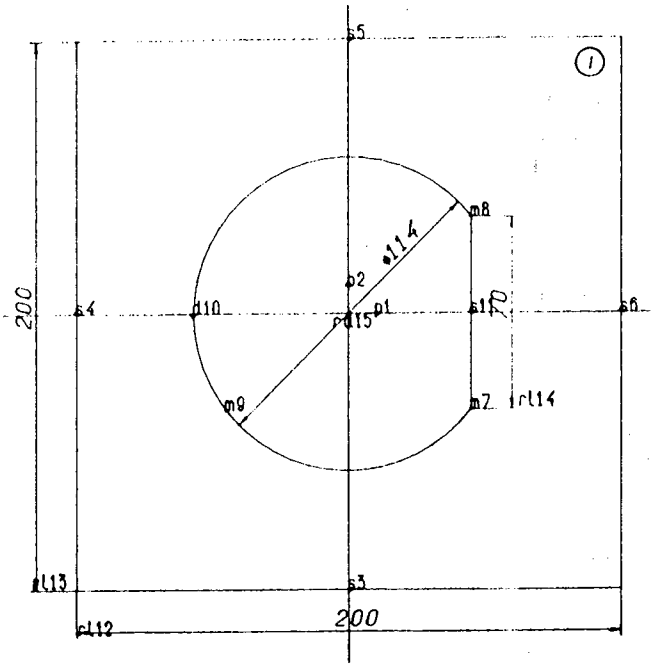


Figure 1. Rough Draft Version of a Sketch

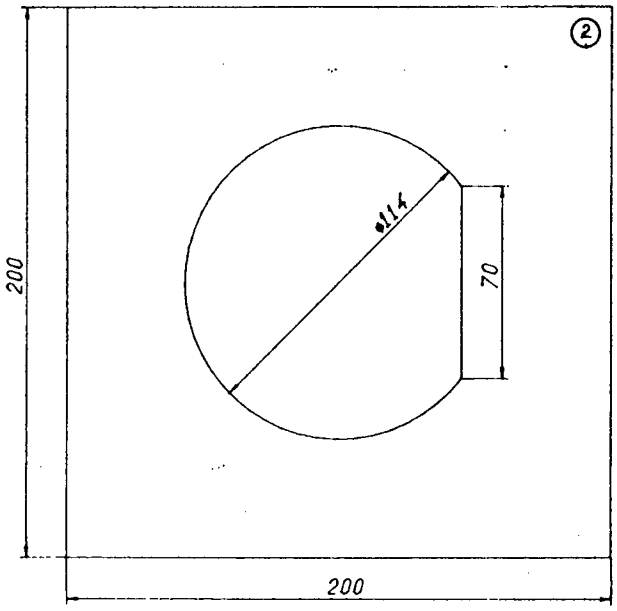


Figure 2. Clean Version of a Sketch

Table 1. List of Canonical Parameters for First 11 Elements of the Sketch

X	Y	F	R	FD
0.00	0.00	0.00		
0.00	0.00	1.57		
-100.00	-100.00	6.28	200.00	
-100.00	-100.00	7.85	200.00	
-100.00	100.00	-6.28	200.00	
100.00	-100.00	-4.71	200.00	
45.00	-35.00			
45.00	35.00			
-45.00	-35.00			
0.00	-0.00	0.66	57.01	4.96
45.00	-35.00	1.57	70.00	

initial angle; and FD is magnitude of the arc. Calculations of the canonical parameters, as well as of the simplest characteristics of the whole entity (for example, the area) can be performed automatically by GRAFIT PPP.

/Operating Characteristics./ [in boldface] The volume of a GRAFIT batch totals approximately 15,000 statements. The SMOG [graphic queueing system] [2] system was used as the basic graphic system for obtaining a hard copy and the SPO GD [graphic dialog software system] [3] as the basic interactive system. The language is functioning at a number of the country's VTs [computer centers] in configurations, when BESM-6, YeS [single series] and M4030 computers are used as a large computer and ARM-R, ARM-M (on an SM-3, M400 base) and "Elektronika 100I" with the "Delta" display unit as an input-output satellite computer.

One manages to accomplish input, editing and output of actual sketches more rapidly by means of the GRAFIT language than by hand. The VTs SO AN SSSR [USSR Academy of Sciences Siberian Branch Computer Center], NGU [Novosibirsk State University] and a number of USSR organizations were involved in developing the language.

BIBLIOGRAPHY

1. Sirotin, V. G., "GRAFIT--SPPP [System of Applied Program Batches] for Automating Planning and Design Operations in Machine Building," in the collection "Problems of Computer Graphics," Novosibirsk, VTs SO AN SSSR, 1982, pp 103-110.

2. "Software Support of Plotters. SMOG: Level 1," Novosibirsk, VTs SO AN SSSR, 1976, 118 pages.

3. Debelov, V. A., Matsokin, A. M. and Chubarev, A. I., "SPO GD--Graphic Dialog Software System," "Problems of Computer Graphics," Novosibirsk, VTs SO AN SSSR, 1982, pp 64-71.

9889
CSO: 1863/69

UDC 681.324

AUTOMATION OF DEVELOPMENT OF SOFTWARE FOR SWITCHING UNITS THAT ARE IMPLEMENTED IN THE FORM OF MULTIPROCESSOR SYSTEMS ON A MICROCOMPUTER BASE

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84 (signed to press 14 Sep 83) pp 41-44, 125

[Article by G. R. Ovchinnikov, Yu. D. Yurakov and A. G. Dmitriyev: "Automation of Development of Software for Switching Units That Are Implemented in the Form of Multiprocessor Systems on a Microcomputer Base"]

[Text] At the present time microcomputers, which in a number of instances are united with multiprocessor and multicomputer control systems for improving reliability and performance as well as for expanding functional resources, are widely used for building switching units in communications engineering. Problems of developing software (PO) occur during the creation of systems like this. Series microcomputers have limited resident means for programming and debugging. As a rule, programming languages, if they're also included in the resident software, are low-level languages and they don't have appropriate analogs in multipurpose computers.

The given circumstance makes impossible the independent debugging and checking of the functioning accuracy of programs, which are written for microcomputers, in multipurpose computers. The advisability of using large computers in the industrial chain for obtaining microcomputer software leads to the development of cross aids that at the present time are available in practically any developed microcomputer system. However, as a rule, the existing cross systems of series microcomputers are oriented towards a single processor configuration and they provide for the debugging process at the level of the command system of the corresponding microcomputer.

The switching units of data transmission systems (SPD) are usually executed in the form of multiprocessor computer complexes (MVK). The software of switching units of data transmission systems has a volume on the order of tens, and sometimes also hundreds, of kilobytes and a comparatively complex structure.

An automated program debugging system (ASO), which allows the following, was created for improving the debugging effectiveness of software that is being developed:

--to improve the writing speed of programs,

--to provide for program verification under the conditions of simulating the operation of hardware that is organized in the form of MVK with a common main memory field,

--to verify the accuracy of operations of applied programs in a complex with an operating system, and

--to provide for the collection of statistical data on the functioning of the computer system.

We'll examine the basic principles of building an automated program debugging system and the results of using it for planning the software of switching units for data transmission systems.

/Structure of a Program Debugging System./ [in boldface] Unification of the basic programming stages--translation, verification of functioning accuracy in a real environment, and correction (if and when necessary)--into a single closed industrial process with united data bases and united software and hardware was a basic requirement that determined the structure of the program debugging system.

At the present time, the demand is growing for similar systems and, therefore, they are being developed by a large number of organizations. The industrial complex for the production of programs with the use of R-technology can serve as an example [1].

The structural components of the automated program debugging system that is being examined (translator, subsystem for text correction and simulation model) are united with common data bases. Control communications are accomplished through a root segment that is common for all programs of the automated program debugging system.

Verification of the operating accuracy of programs under static and dynamic conditions is a definite task of the simulation model--the basic part of an automated program debugging system. Verification like this is conducted most effectively in the interpretation mode, since in this case errors in the program that is being debugged do not lead to an accidental halt of the model as a whole. In addition, the operator has the opportunity to actively intervene in the simulation process.

When developing an interpreter, it's important to select the level of the input language on which the verification depth of the algorithms that are being simulated depends. Emulation, i. e. interpretation at the command system level, is used in an automated program debugging system. The emulator makes it possible to build a simulated model that in the most complete manner takes into consideration the structure of the actual entity, to reduce the memory capacity for storing programming modules that are being debugged, and to develop programs both in a high-level language (if there are appropriate translators) and at the command system level.

During simulation it's possible to use active control of the algorithms that are being simulated or to interpret them. It's more efficient to use active control for obtaining statistical dependencies, since in this regard the model's performance turns out to be substantially higher than when emulating the same programs. Inasmuch as the model that is under development is multi-functional, both control methods are stipulated in it.

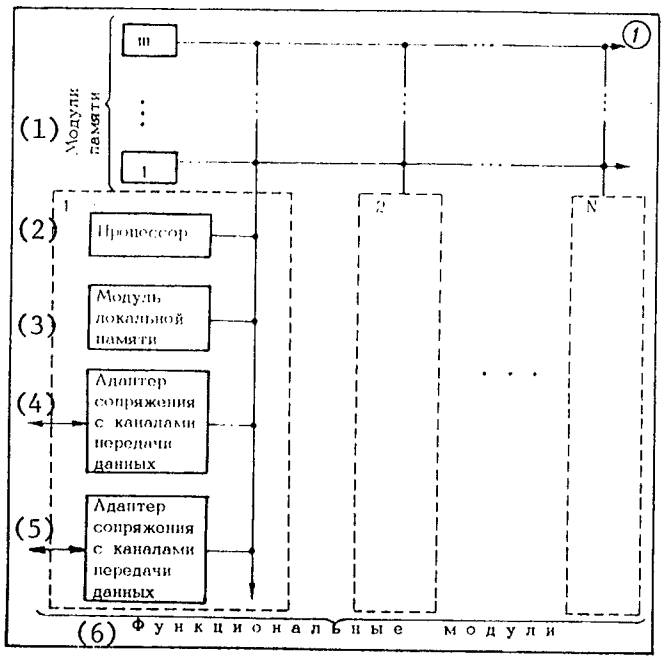


Figure 1. Generalized Structure of MVK [Multiprocessor Computer Complex]

Key:

1. Memory modules
2. Processor
3. Local memory module
4. Adapter for interface with data transmission channels
5. Adapter for interface with data transmission channels.
6. Functional modules

The emulator of an MVK with a common main memory field forms the basis of the simulation model. The generalized structure of an MVK is cited in figure 1. The MVK consists of a set of functionally oriented modules, which are united

for joint operation with the common main memory field that is broken down into a number of memory modules, and for each of which there is an independent access channel. The functional model consists of a processor, a local memory module, and a number of external devices; for example, adapters for interfacing with the data transmission channels. The structure of the hardware (number of processors, capacity of local and common memory) is prescribed by the user and can be altered during the process of operation.

Generation of external effects in the model and execution of algorithms of software that is being simulated can be accomplished in the active control mode.

The possibility was stipulated for triggering programs both by time traces and by events, and that is used for organizing communcations with the external medium.

The structure of an automated program debugging system is cited in figure 2.

Control of the model can be accomplished either in an interactive mode by means of a video terminal unit or in a programmed manner.

Programming is accomplished in the same control statements that are used when operating in the interactive mode. A program can be generated beforehand or at the time the model is operating in the interactive mode. Special statements are stipulated for organizing cyclical and branching programs.

At the present time there are no specially developed programming languages (YaP) for microcomputers that are considered at the algorithmic level of the functioning feature of multisystems. However, programming languages that are oriented towards writing programs for microcomputers have already been created; for example, PLM. In a number of instances, existing programming languages are being adapted for these purposes; for example, there is a version of microPASCAL language for programming parallel multisystems that are built on the base of microcomputers [2]. A similar example of using a universal programming language of multisystems is cited in [3].

The following situations are taken into account when selecting the programming language for an automated program debugging system. It's not advisable to use an autocode as a programming language, since different microcomputers can be used in different modifications of switching units. In addition, a transition from one family of microcomputers to another can occur in the improvement process. A programming language must provide the possibility for obtaining optimum object programs. It's advisable to use programs, which are written in a programming language, for building different level simulation models in multipurpose computers.

Use of the original version of a language makes it possible to most fully consider the building features of each specific control computer complex. However, in this case, the use of multipurpose computers is hindered in the debugging process. When developing the original translator, it's necessary to implement all phases of compilation in full capacity.

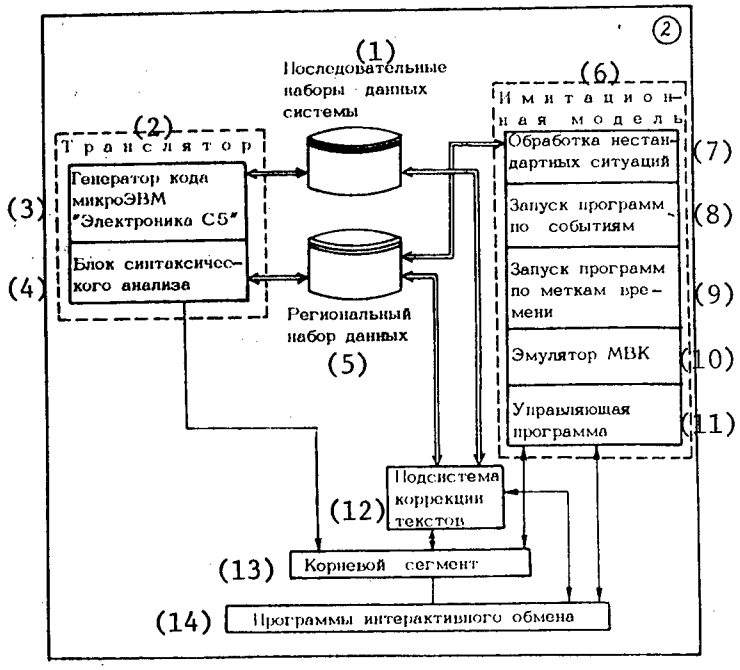


Figure 2. Structure of an ASO [Automated Program Debugging System]

Key:

1. Sequential data sets of a system
2. Translator
3. Code generator of "Elektronika S5" microcomputer
4. Syntactical analysis unit
5. Regional data set
6. Simulation model
7. Processing of nonstandardized situations
8. Triggering of programs by events
9. Triggering of programs by time traces
10. Emulator of MVK [multiprocessor computer complex]
11. Control program
12. Subsystem for correcting texts
13. Root segment
14. Interactive exchange programs

A subset of the PL/1 language, which is expanded by a number of subprograms for providing multiprocessor and multiprocessing processing of jobs, is used as the programming language in the automated program debugging system. In the first place, a solution like this makes it possible to organize double use of one and the same programming modules: both in the simulation models that are implemented on the base of a YeS [single series] computer and for obtaining programs in the codes of one microcomputer or another. Secondly, it makes it possible to provide for independent debugging of the programming modules in the YeS computer , while using the entire high-power debugging and diagnosing equipment that is available in the software of the given series of computers. In the third place, the use of a standard compiler of PL/1 for conducting syntactical analysis and control makes it possible to substantially reduce the volume of its own developments. Fourthly, it's necessary to consider a knowledge of the given language by a sizable contingent of programmers, and that reduces the required volume of instruction.

A basic shortcoming of a solution like this is the impossibility of taking into consideration the building features of a control computer complex, which are the switching units of data transmission systems, and it is partially eliminated by the introduction of subprograms that provide for the contact of applied programs with the operating system of multiprocessor computer complexes.

Contact of the subsystems with the unified system and interaction with the external medium are accomplished at the level of data sets, and for the operation with which the subsystem for correcting texts is included in the composition of the automated program debugging system. This subsystem provides for preparation as well of the regional data set for operating the translator and the simulation model.

/Simulation Model./ [in boldface] As was noted already, a simulation model comprises the basis of the automated program debugging system. A number of statements, which either are inputted directly in the interactive mode during simulation or are prepared beforehand in any sequential set and then inputted to the model in the program control mode, was stipulated for controlling the model.

The /basing statements/ [in italics] make it possible to determine the area of operation of the statements, which are examined below, in the basic memory of the MVK: either as the local memory of one of the modules or as one of the modules with a common memory that is accessible to all functional modules.

The /statements for displaying and altering the system's status/ [in italics] make it possible to display or alter the status of any module of the system. Statements for displaying and altering the contents of the system's basic memory are included in this group.

The /statements for operating with the data sets/ [in italics] make it possible to input data to the basic memory of the system that is being simulated from the sequential data sets, as well as to transfer the contents of the basic memory to the sequential data sets. A flexible magnetic disk unit

(NGMD), with which actual complexes can be fully configured, is simulated in the system that is being examined. Both sequential and library data sets can be organized in the NGMD. There is a statement that implements direct access to any data set in a flexible disk model. In a simulation model it's possible to have a data copying set in which all information, while being displayed on a video terminal, is recorded in the event of a job under the proper conditions.

The /statements for controlling the operation of the functional modules/ [in italics] make it possible to stop or trigger any module of the MVK. There is a statement that provides for the over-all start up of the system and the prescribed number of instructions that must be performed in a given simulation cycle. A proper and sufficient level for displaying the simulation process is of great importance when debugging programs. In the system that is being examined, emulation can be performed with a display or without it. The display can be accomplished either at the command system level of the microcomputer or at the programming language level. In the first case, the following parameters appear on the screen for each instruction: instruction counter, instruction, effective address of the operand, contents of the effective address, and the results flag. A display of the instructions that are being executed in the programming language are stipulated in the simulation model. In this regard, the program text in the programming language appears on the screen, and a cursor indicates the statement that is being executed at a given moment of time. In addition, the contents of all registers of the module, in which the given program and the execution results of the operation are performed, are displayed. The program text for display is prepared by the translator of the automated program debugging system in the regional data sets.

The /program debugging aids/ [in italics] were stipulated in a broad set. In each functional module, it is permitted to assign instruction counter halts (up to eight enter and read halts). In addition, in each program that is being debugged it's possible to introduce special instructions by which the so-called program halts are executed. At the moment of its occurrence and depending on the type of instruction, the message concerning the halt either appears on the screen of the video terminal, is stored in the data copying set, or the user's program is triggered at this moment. For example, this program can collect additional information concerning the halt or simulate one algorithm or another of the MVK in the active control mode. If an output to the screen is provided at the moment of the program halt, then the system can switch to the program control mode and execute in this mode the program that was assigned earlier.

The /statements for operating with systems time/ [in italics] make it possible to display and correct the system's operating time.

The /statements of the initial initiation/ [in italics] of the system, in the operating process of which loading is provided for all modules of the nucleus of the OS [operating system], editor, and methods of access to the data for the editor, are included in the simulation model. Following operation of the initial initiation statement, it's possible to accomplish further loading of software according to the actual functioning protocols of a specific MVK.

/Language and Translator of the System./ [in boldface] In the programming language that is being used, the use of all arithmetic and logic operations was permitted with one limitation: multiplication and division were permitted only for a whole number. Addition procedures and functions according to modulo two and logical shifts to the left and to the right were included in the composition of logic operations.

The use of conditional statements, GOTO statements and noniterative DO groups was permitted in the programming language. The insertability of the groups was not limited.

Access to the subprograms is accomplished by means of a CALL statement. The transmission of parameters is accomplished through data structures that are common for the subprograms. Each subprogram, except the basic one, can have additional input points.

Identifiers and variables with an index are used as markers in the programming language. Language statements that are marked by the translator aren't perceived and, when using programs that are written in the programming language and in different simulation models, that makes it possible to input additional functions, which will not be reflected in the objective modules that are received in the microcomputer codes.

In the dispatching system of the MVK that is described in [4], a great deal of attention is devoted to data structures that can be organized both in the local memory of the microcomputer and in the common memory of the MVK. Therefore, the structures are assumed as a basis for describing data in the programming language. The based structures make it possible to operate with the virtual memory of the MVK. It's permitted to use arrays in the structures, however, the number of indexes in an array is limited and that is connected with writing optimum programs in microcomputer codes.

The testing trials that were conducted showed that object modules, which were obtained following translation, are 10-20 percent longer than similar program modules that are written directly into the microcomputer codes.

/Practical Results./ [in boldface] The examined automated program debugging system was used by the authors for debugging the subsystem of centralized process dispatching, which is included in the operating system of switching units of data transmission systems and is designed for organizing the computer process in multiprocessor computer systems that operate under real time conditions. Look-ahead operating conditions make it possible to implement real time conditions without using synchrointerrupts.

The distribution of all kinds of resources is organized at the level of processes that are operating units in the system. The interaction of applied programs with the subsystem that is being debugged is accomplished by means of systems statements: GET RESOURCE, RELEASE RESOURCE, OBTAIN CURSOR, OBTAIN ACCESS TO SYSTEMS COMPONENT, RELEASE SYSTEMS COMPONENT, DELAY PROCESS, TRIGGER PROCESS WITH DELAY IN TIME, TRIGGER SYNCHROJOB and others. In [3] one can

become acquainted in more detail with the organization of the subsystem of centralized process dispatching.

The development of a subsystem of centralized process dispatching was conducted in several stages. During the first stage, macroinstructions were developed on the basis of which program modules of systems statements were assembled during the second stage. Both the majority of macroinstructions and all systems statements were written in the programming language and each of them was debugged either independently with the use of regular means of the YeS operating system or with the use of a simulation model that is included in the automated program debugging system. Debugging of individual modules was done under uniprocessing conditions. A test example was developed for each systems statement.

Comprehensive debugging of all systems statements under dynamic conditions was done following independent debugging.

A dual-processor computer system with a common main memory field, in which a number of job queues, a processor queue, and a number of resource queues and general-purpose tables were written, was created on the base of an automated program debugging system. At the beginning of the operation, several information process domains were generated for each job queue and that made it possible to parallel the execution of a number of single-type jobs. Subsequently, the number of processors in the system was increased.

The use of independent debugging made it possible to reveal practically all of the serious errors.

For optimizing the algorithms of the operating system and the collection of statistical data about the operation of the data transmission systems's switching devices as a whole, the model was started in an automatic mode, for which generators and data receptors were worked out. Examining the results of statistical modeling is beyond the scope of this article.

The program volume of the subsystem of centralized process dispatching generally totalled about 8K bytes.

At the present time, all systems statements of the subsystem are included directly in the software of the automated program debugging system and that provides an abstract computer model. The automated program debugging system with the subsystem of centralized process dispatching and the initial loading programs is used for developing the applied software of the switching units of data transmission systems, as well as for further development of subsystems that are included in the operating system.

/Conclusion./ [in boldface] The proposed automated program debugging system makes it possible to develop and debug software for multiprocessor systems, which are built on the base of series microcomputers with the "Elektronika S5" command system, in the interactive mode. The automated program debugging system can be used as well for simulation modelling of algorithms for processing information in the switching units of data transmission systems.

The automated program debugging system is executed in a YeS computer, supported by the version 6.1 YeS operating system, and uses a YeS7029 display set for its operation. The total volume of the load modules amounts to about 80K bytes. The automated program debugging system is implemented with an overlay structure for economizing the main memory of the computer.

The authors express gratitude to O. V. Dyudyakov who took an active part in developing programs of the subsystem of centralized process dispatching.

BIBLIOGRAPHY

1. Vel'bitskiy, I. V., Khodakovskiy, V. N. and Sholmov, L. I., "Industrial Complex for the Production of Programs for YeS and BESM-6 Computers," Moscow, STATISTIKA, 1980, 262 pages.

2. Fulton, S. and Vuffen, R., "High Level Language for Developing Real-Time Systems Programs," ELEKTRONIKA, Vol 53, No 26, 1980, pp 51-57.

3. Fuller, S. Kh., Usterkhut, Dzh. K., Raskin, L. and others, "Multimicroprocessor Systems. Review and Example of Practical Implementation," TIIER, Vol 66, No 2, 1978, pp 135-150.

4. Antoshevskiy, V. S., Ovchinnikov, G. R. and Yurakov, Yu. L., "Building Software of the Switching Center of Batches. TEKHNIKA SREDSTV SVYAZI [Communications Facilities Engineering], "Line Communications Engineering" series, No 8(4), 1981, pp 31-37.

9889
CSO: 1863/69

UDC 681.326.3

REALIZATION OF FORTRAN IN INTELLIGENT TERMINAL

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84 (manuscript received 13 Dec 83, after completion 31 Jan 84) pp 57-61

OSTROUKHOV, D. A. and GULINSKIY, P. Ya.

[Abstract] Program debugging in higher-level languages by a microprogram computer acting as collective intelligent terminal has been realized in FORTRAN-77, with P-FORTRAN as intermediate language, so as to allow for transfer of debugged programs to other computers such as an "El'brus" multiprocessor complex. This realization ensures filtration of programs for adequate control of deviations from the standard input language and facilitates operation with several formats of problem data introduced either before or after translation. It also provides for hookup of interactive debugging software in the input language. The realization follows a two-level scheme and is supported by hardware in the collective intelligent terminal. The filtration problem has been solved by verification not of the program as a whole but of its agreement with the description in the input language during each specific run. Verification includes static and dynamic tests, the filtration program being executed in FORTRAN-77 according to ANSI-X.3.91978 specifications. Figures 3; references 13: 5 Russian, 8 Western (2 in Russian translation).
[68-2415]

UDC 681.3.02:519.682.2

DECOMPOSITION OF REQUIREMENTS IN 'DESIT' LANGUAGE

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84 (manuscript received 26 May 82, after completion 6 May 83) pp 61-66

BASIN, Yu. A.

[Abstract] A possible approach to construction of problem-oriented controlling monitors for system simulation and design is shown, this method being

implemented conceptually and structurally in the DESIT (DEkompozitsiya SIstemnykh Trebovaniy = DEcomposition of SYstem Requirements) language. This language, developed specially for describing the functioning of systems with Backus-Naur syntax, makes it possible to describe the interaction of system components with maximum diversity of interaction structures while partitioning the designed system into its operation component and object component. The language provides also for automatic verification of completeness and consistency as well as for introduction of adequate code-generating monitors. The design procedure involves decomposition into processes interacting within the system and specification of their interaction, which is demonstrated here on two examples, followed by verification with the aid of connection and attainability matrices containing a logic "1" each. The code of a controling monitor consists of two programs, a PROCESS INITITIATION first program and an INTERROGRATION ANSWERING second program, controlled by a supervisor. Functional decomposition is effected in some virtual medium of system evolution, this medium being describable with drivers in accordance with abstract data types incorporated in the PASCAL language, whereupon actions are described with the aid of module identifiers. In the DESIT language it is possible to simulate the control of unprogrammed systems as well as the control of programmed ones. Figures 1; references: 6 Western (3 in Russian translation).
[68-2415]

APPLICATIONS

UDC 681.51

COMPUTER-AIDED INSTRUCTION SYSTEM ON AN ARM/"NAIRI-4" COMPUTER BASE

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 5, Sep-Oct 84 (signed to press 2 Mar 83) pp 111-114

[Article by A. O. Vardanyan and B. V. Pyrinov: "Computer-Aided Instruction System on an ARM/"Nairi-4" Computer Base"]

[Text] The AOS-ARM/"Nairi-4" computer-aided instruction system was developed and assimilated in series production on an ARM/"Nairi-4" minicomputer base at the "Elektron" (Yerevan) production association jointly with the Novosibirsk Institute of Railroad Transportation Engineers (NIIZhT) and an affiliate of the All-Union State Design-Engineering Institute for the Mechanization of Accounting and Computing Operations.

The AOS-ARM/"Nairi-4" is a multivideo terminal system that is program compatible with similar complexes on an M400, SM-3 base under ARM [automated work place] conditions, and it is designed for the automation of operations during the design of radio-electronic equipment and machine building products. The system is used under AOS [computer-aided system] conditions during the independent study of various academic disciplines on instructional programs (OP) that are drawn-up by users.

In the article, a description is given of the system's hardware, software and main ideas that are assumed as a basis for the language of the authors of the instructional programs.

The following are included in the hardware that support the AOS conditions:

--main circuit switch (KMK),

--two group control units (BGU),

--two display unit switches (KD), and

--32 RIN-609 ("Videoton-340" display unit analog) video terminals.

A processor with a 64K main memory, a timer unit, an operator's console with a "Konsul-260" typewriter, an FS-1501 tape transmitter, and a PL-150 paper-tape punch are used under AOS conditions in addition to the indicated devices.

The structural diagram of the AOS-ARM/"Nairi-4" is cited in the figure.

The KMK, BGU1 and BGU2 are placed in a rack of the ARM/"Nairi-4" computer's processor; the KD1 and KD2 are located in separate ShAU1 and ShAU2 [lecture hall cabinet] racks. The ShAU racks can be placed in different lecture halls. The maximum distance between the BGU and the ShAU is 60 meters. The video terminals from the ShAU can be located at a distance up to 16 meters. A structure like this makes it possible to conduct lessons not at computer centers, but in special lecture halls.

The video terminals can operate under one of the following four conditions:

--processor input,

--processor output,

--input by direct access, and

--output by direct access.

The exchange of information between the video terminals and computers with large bloch (up to 960 characters) occurs under direct access conditions.

Processor input is accomplished in the following manner.

Each BGU has its own time-pulse generator and 4-bit time-pulse counter. At the time of polling $t_{оп}$, all 16 display units are in turn connected to the computer by means of the time counter. When the transmission of a character from the keyboard to the computer occurs (when one of the keys is depressed), the code of the character is entered in the output register of the display unit and the keyboard is interlocked. During polling of a given display unit, the code of the character that is being inputted from the output register through the KD enters the BGU and is entered in the input data buffer register (RDBVV). If the data entered the BGU during polling or prior to it, then the time counter stops (its contents always correspond to the number of the display unit that is being polled), and interrupt and servicing of the appropriate display unit are accomplished. In other words, the processor reads the contents of the RDBVV and enters them in the appropriate section of the computer's OZU [main memory]. Servicing time is $t_{об}=1\div3$ microseconds and polling time is $t_{оп}=1\div2$ milliseconds, i. e. $t_{оп}\ll t_{об}$. [sic],

Following servicing of the display unit, the keyboard will be released according to the algorithm, the time counter will begin to operate and the process will continue.

There are a 4-bit display number register (RND) and an 8-bit output buffer data register (RDBV) in each BGU for processor output.

Output begins with an entry in the RND following which the readiness of the display unit is verified for receiving information. If it's ready, then the processor enters the code of the information that is being outputted to the RDBV, and further through the KD it enters all 16 display units, but is entered in the main memory of the one of them the number of which is fixed in the RND. Following this, the BGU receives a signal from the display unit concerning the reception of information and it generates an interrupt signal. This means that the processor can output the next byte of information (character).

When there is an output of large blochs, it's convenient to use direct access conditions. The bloch that is being outputted is stored in the processor's main memory. The readiness of the display unit to receive information is verified just as under processor output conditions. The initial address of the bloch that is being outputted is entered in the address register (RGA) that is located in the BGU. The BGU organizes a byte-by-byte reading of data from the processor's main memory and the entry of this information in the RDBV, and further, just as under processor output conditions, the data enter the appropriate display unit from the RDBV. Following the output of each character, a unit is added to the contents of the RGA. Having recognized the KT (end of text) code, the BGU informs the processor of the termination of output.

It's convenient as well to execute the input of large blochs by direct access. In the independent mode of the display unit, the information that is being inputted is entered in its main memory with a display on the screen and edited, following which the "data transmission" key is depressed. In this regard, the display unit shifts from the INDEPENDENT mode to the TRANSMISSION mode and the KT code is entered in its main memory. The characters enter the RDBVV during polling of a given display unit (as under processor input conditions). From the moment the first character enters the RDBVV, the time counter is stopped until the termination of direct access conditions. With entry of the interrupt signal, the processor enters in the RGA the initial address of the allocated area of the main memory where it's necessary to place the bloch that is being inputted.

During input by direct access, The BGU organizes data transmission from the RDBVV to the processor's main memory. A unit is added following the entry of each character in the RGA. Following entry of the KT code in the RDBVV and its entry in the processor's main memory, the BGU informs the processor with an interrupt signal of the termination of direct access, and the display unit shifts in a diagram manner to a communications mode with the computer.

The communications length of 60 meters between the ShAU and the BGU is provided by MS1 (transmitter) and MS2 (receiver) microassemblies. It's possible to increase the distance up to 500 meters, if the appropriate receivers and transmitters are replaced with special T-interface receivers and transmitters.

Two programming complexes provide for functioning of the AOS:

--basic time-sharing software (BPORV) that was developed at the Yerevan affiliate of VGPTI [All-Union State Design-Engineering and Industrial Institute] for the Mechanization of Accounting and Computing Operations, and

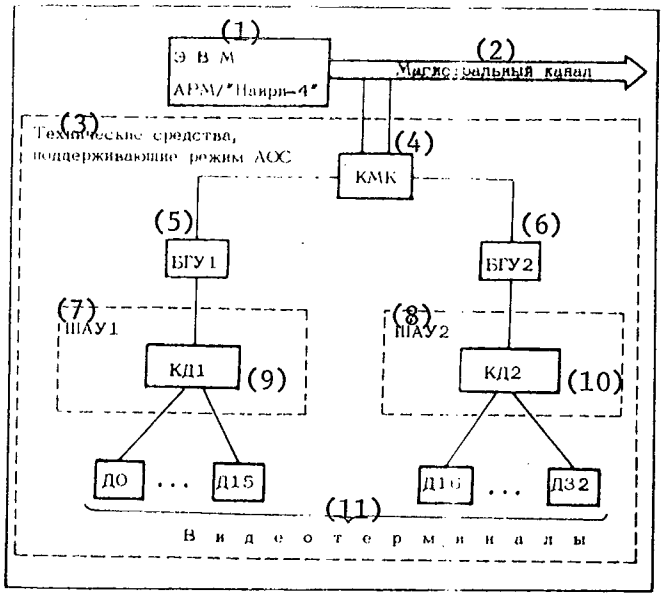


Figure 1. Structural Diagram of AOS-ARM/"Nairi-4"

Key:

1. ARM/"Nairi-4" computer
2. Main circuit
3. Hardware that support AOS conditions
4. Main circuit switch (KMK)
5. Group control unit (BGU1)
6. Group control unit (BGU2)
7. Lecture hall cabinet rack (ShAU1)
8. Lecture hall cabinet rack (ShAU2)
9. Display unit switch (KD1)
10. Display unit switch (KD2)
11. Video terminals

--basic instructional program software (BPOOP) that was developed at NIIZhT.

The BPORV makes it possible to execute one or several instructional programs simultaneously at all video terminals; an input-output control system, a timer servicing program and a number of other subprograms are included in it.

The BPOOP is designed for translating and executing instructional programs, as well as for controlling the progress of instruction on the part of the instructor and the students. The following are included in it:

--instructional program translator,

--program for processing the instructor's directives (ASSISTENT),

--program for processing the student's commands, and

--instructional program interpreter.

The first three programs interact directly with the users and their entire operation is performed in accordance with directives that are submitted in a natural language. The interpreter executes the instructional programs during instruction.

The instructional programs are programmed in the POLINOM-80 language. The instructional program translator and the ASSISTANT program are programmed in this language as well.

The POLINOM-80 language is oriented towards the small computer. Many computer operations here are performed by default and, therefore, the instructional programs in the computer implementation are brief. The language itself and the instructional programs are suitable for various expansions and the connection of modules that execute any operations within the limits of a computer's technical resources.

Adaptive language features take into consideration the preparedness of the student, the complexity of the instructional material, the nature of allowed errors, and the operating rate.

The instructional programs are programmed according to a diagram with multiple branching: an unlimited number of continuations can follow after each student's response. There also can be linear programs that are being used as monitoring ones.

The text of an instructional program is divided into pieces of information that correspond to certain portions of the instructional material. A piece of information has two parts:

--the text proper of a piece of information where the instructional material is presented and one or several questions are generated, and

--standard responses with commentaries on them that are programmed beforehand.

Texts of the pieces of information and standard responses are entered in the computer by separate blochs. The instructional program interpreter keeps both blochs in sight simultaneously.

Both the text of a piece of information and the commentaries on standards are a sequence of characters and instructions of the POLINOM-80 language. The instructional program interpreter scans this sequence and all encountered characters are outputted to a screen, while not requiring any instructions whatever for this. If an instruction (that occupies one or two bytes) is encountered, then control is transferred to the appropriate module for its execution, and then the output of characters continues again. The absence of print statements in an instructional program noticeably abbreviates it, since this is the most frequently encountered statement.

The analysis instructions (presently there are eight of them) are the largest in the number of executable operations. Each one of them:

--compares a response with the standard (standards) by one method or another,

--conducts registration of errors,

--prepares standard or previously prepared commentaries (of the CORRECT--INCORRECT type) for presentation,

--provides for transition to presenting a commentary, and

--following a commentary, prepares the return to a continuation of the piece of information or its partial repetition (requery).

It's sufficient to enter a two-byte instruction in an instructional program and to provide each standard with a three-bit (binary) flag (value) for executing all these operations.

Analysis of the response is conducted for precise agreement with the standard or for being within the numerical range, as well as for agreement with the standard when there is a deviation of 1-2 percent.

The standards themselves can be either written in the instructional program or computed in the course of operation with regard to the student's individual data. A module in PDP-11 language, which is connected to the instructional program by a separate instruction, must be programmed for computing the standards (numerical or character). The maximum number of modules in one instructional program is 255. Thus, there are practically no limitations of any kind on the method for computing the standards. The analysis methods can be developed easily as well.

We'll cite a small example from which the structure of a piece of information and the sequence of the computer's operations will be evident.

```
READ--P.2.1.#TP

WHAT--IS--THE--NAME--OF--          o3 ePIECE OF INFORMATION
THE--BASIC--UNIT--OF--THE--        k CORRECT.  #KS
INSTRUCTIONAL--PROGRAM?            k YOU--MADE--AN--ERROR,
#ChF15#AE15#ID12                     RESPOND--AGAIN:  #KS
```

The text of the piece of information is cited on the left here and on the right is the standard, which is marked in the front with the letter "e" and has the value three, as well as two commentaries in the event of a correct or incorrect response (they are marked with the letters "k").

The instructions in the text are provided with the flag #, and in the computer entry with a unit in the master bit of a byte.

Having begun the presentation of a piece of information, the computer will depict on the screen the message READ P.2.1. WHAT IS THE NAME OF THE BASIC UNIT OF THE INSTRUCTIONAL PROGRAM? and it will stop at the instruction #ChF15 (read the phrase in the 15th cell of the student's main storage region). The place from which it's necessary to repeat the presentation of the piece of information, if an incorrect response enters, will be fixed by the instruction # TP (requery point).

The response is verified by the instruction # AE15 (analyse the response in the 15th cell according to the standards).

If there is a correct response, estimate 3 provides for output of the first commentary CORRECT and presentation of the piece of information will continue, but only instruction # ID12, in accordance with which transition to piece of information 12 will occur, is there.

If there is an incorrect response, a second commentary will be output in accordance with value three, following which a portion of the text of the piece of information will be repeated: YOU MADE AN ERROR; RESPOND AGAIN: WHAT IS THE NAME OF THE BASIC UNIT OF THE INSTRUCTIONAL PROGRAM?

Both commentaries here conclude with the instruction # KS (end of message), but they also can end with a transition and then a return to the text of a given piece of information doesn't occur.

An error that is allowed during a response and the very fact of presenting a piece of information are fixed in the counters that are not visible to the programmer; when there is some other value (not equal to three), loading of the counters does not have to take place.

From the example it's evident that the instructions of an instructional program occupy some space in a computer's memory, and the even greater consolidation of a program is possible only at the expense of texts. The dictionary of repeating phrases serves this purpose. Words, combinations of words or their parts and, finally, simply a program's linear segments that consist of characters and instructions are entered in it. Access to the dictionary is accomplished by the instruction #VI-α (output phrase with the number α) that is entered in the necessary space of the text of a piece of information1 or commentary. The total number of phrases in a dictionary is up to 255. Experience shows that they make it possible to reduce an instructional program by approximately one-third.

The use of subroutines provides a substantial reduction as well. A subroutine is a small, branched instructional program that consists of a number of pieces of information to which access is possible from different locations of the basic program. For example, the necessity for this occurs during an explanation of the preceding material, a lack of knowledge of which can be revealed at different stages of the student's operation. Subroutines are inevitable as well in large instructional programs that control course planning.

The total impact from using all means for program reduction can be described by the following numbers. One of the instructional programs that controls the execution of a course plan at NIIZhT occupies a memory capacity of nearly 5.5K and interacts with a student over the course of 6-8 hours. Another one, which is more laconic, occupies 10K and works 25-30 hours with a student.

Its speed is a considerable factor for the possibility of creating an AOS on the base of a small computer. Of course, the multiplicity factor of a program accelerates the servicing time of a single subscriber. But this is still insufficient to provide an acceptable reaction time to a student's response. Processor input and output spoke a decisive word here. Analysis of the accuracy of a response can proceed following the input of each character. In many instances, it's possible to recognize a correct response by one to two letters. In the AOS-ARM/"Nairi-4" it's easy to organize the printing out of a response like this and not to take time away from the student and the computer. An incorrect response is rejected during the earlier input stage, the computer is released from reading and analysing unnecessary information, and the analysis itself is also considerably simplified. While recognizing and printing out individual words, case endings and so forth, it's possible to receive and analyse in increments the more complex responses that are impossible to identify by one to two letters. Accordingly, a response is generated jointly by the student and the computer. Thus, ACCORDING TO THE ESTIMATE OF THE STANDARD will be the correct response to the question ACCORDING TO WHICH COMPUTER INDICATOR IS THE NECESSITY FOR ERROR REGISTRATION DETERMINED?. It's sufficient for the student to set up the underlined letters and the computer will add the remaining ones. It's clear that there is an element of prompting here, but not so considerable. In return, the analysis program is reduced many times, the entire operation is accelerated, the reaction time to a response is reduced, and the memory's main storage region in which it's necessary to read the student's response is considerably reduced.

Everything that was said doesn't mean that the POLINOM-80 language doesn't allow other forms for analysing responses. This is easily feasible, but limitations on the number of simultaneously operating terminals can occur.

The adaptive means of the POLYNOMIAL-80 language are practically the same as those in other instructional systems [1-3], but they differ in the brevity of program implementation. A complexity indicator P is established according to each piece of information. The simplest pieces of information have P = 0 and the most complex ones have P = 7. Accordingly, all students have a preparedness indicator p. The poorest ones have p = 0 and the strong ones have p = 7. A piece of information is presented to a student only if $p \geqslant P$, i. e. if it's within his abilities. Otherwise, an equivalent piece of information, which is

stated in a more detailed and accessible manner, is presented. While using the indicators p and P (again without additional instructions in the instructional program), the instructional program interpreter selects a piece of information that is within his abilities. This selection proceeds during any transitions to a new piece of information. Thus, not every student will receive the 12th piece of information in accordance with instruction # ID12 and a simpler piece of information will be presented to some of them, but if it is too complex also, then the computer searches for one that is within their abilities. Up to seven such sequential transitions are stipulated in accordance with a string of equivalent pieces of information, i. e. eight levels of complexity of the instructional material are possible. In instructional programs on course planning, it's suitable to use indicators P for partially and completely automating a calculation. For example, P = 0 ÷ 4 can signify the ordinary levels of complexity, P = 5 ÷ 6 signifies partial and P = 7 signifies complete automation of a calculation.

Each piece of information has a time norm for studying it and, when there is too much time, the interpreter presents an equivalent piece of information.

In our opinion, the problems of trouble-shooting the typical errors of a student and his reaction to them have been solved successfully in the system. Replication equipment exists for this. Replication has two texts. The first text (replication A) is an ordinary reaction to an error. For example:

INCORRECT. # KS

The second text (replication B) contains a reaction already to a specific type of error. For example:

RECALL--THE--METHOD--FOR--INTEGRATION--BY--PARTS. # ID78.

The replication is built in to the commentary at different places of a program where a given error is "caught" by the instruction VRk (k is the replication number). Replication A is output the prescribed number of times (two to three times) and then replication B is dumped one time, and the presentation counter of replication A is erased as well. All this makes it possible to react easily (without the expense of the place in a program) to an error and then to provide additional material for study, to reduce the preparedness indicator, and so forth.

It's a sound practice to use two kinds of replication for reviving a dialog: GOOD, and sometimes WELL DONE, are used with all successful responses; NO, NO, and at times, PLEASE DON'T HURRY! are used with unsuccessful responses.

A prototype of the described AOS was implemented at NIIZhT in a "Nairi-K" computer and it has already been used for 5 years in the instructional process of a "Bridge Design" course [4]. Experience showed that instruction in the design process is a very promising area of application for AOS. The plan manages to be replete with new contents, to introduce elements of SAPR [computer-aided design systems], to improve the study of instructional material by all students, to rid them of a number of routine computations, to dramatically

reduce (up to one-third and one-half) the execution time of an operation, and to heighten interest towards the subject that is being studied.

BIBLIOGRAPHY

1. Dovgyallo, A. M., "Dialog of a User and a Computer. Bases of Planning and Implementation," Kiev, NAUKOVA DUMKA, 1981, 232 pages.

2. "Computer-Aided Instruction Systems on a Computer Base," edited by A. F. Chernyavskiy, Minsk, BGU [Belorussian State University imeni V. I. Lenin], 1980, 176 pages.

3. Kuznetsov, S. I., "'Sadko'--A System for Computer-Assisted Dialog and Collective Instruction," VOPROSY KIBERNETIKI, "Human and Computer Instructional Systems" series, No 60, 1979, pp 150-160.

4. Pyrinov, B. V., "Systematic Instructions for Calculating Reinforced Concrete Flexible Components of Bridges in the Form of a Dialog With a "Nairi-K" Computer," Novosibirsk, NIIZhT, 1982, 28 pages.

9889
CSO: 1863/69

- END -